# The Rheology of Nanoparticle Additives: An Investigation Utilizing Mesh Free Methods

Jonathan P. Kyle

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts & Sciences

Columbia University
2014

ABSTRACT

The Rheology of Nanoparticle Additives: An Investigation Utilizing Mesh Free Methods

Jonathan P. Kyle

This dissertation applies mesh free computational methods to investigate the rheological impact of arbitrarily shaped nanoparticle additives in shearing interfaces. Specifically, Smoothed Particle Hydrodynamics is used for its flexibility in modeling moving fluid-structure interfaces, the ability to model non-Newtonian fluids, as well as having the capability to add any additional physics deemed appropriate. With this modeling technique, a sufficient theory for the non-Einstein like rheological modification seen with certain nanoparticle additives is achieved based on surface tension effects between the additives and solvent. Computational results are compared with experiment resulting in good agreement.

# Table of Contents

# List of Graphs, Images, and Illustrations

# Acknowledgements

I would like to thank my advisor, Professor Elon J. Terrell, for his unwavering support since my undergraduate studies. I would like to thank the members of my committee for offering their invaluable insight and expertise. I would like to thank my fellow doctoral students - those who have moved on, those still deep in research, and those just beginning - for their support, friendship, and all around good humor. I would like to thank all the staff in the Mechanical Engineering department, for making sure I progressed smoothly in my studies. I would like to thank all of my professors, for sharing their knowledge and love of learning with me. I would like to thank all my friends, for their support and good cheer. I would like to thank my labmates, for offering constructive feedback with my research and for always being by my side. Last, but certainly not least, I would like to thank my family - it is on all your shoulders that I now stand.

# Dedication

To Sylvia A. Paul Ellis and Earle F. Kyle, Jr., my parents and my foundation.

To Kanika Moxley, my sister and show of strength.

To Justus Simmons and Jahnarie Simmons, my nephews and future.

# Introduction: Overview of Research

The objective of this research is to apply mesh free computational methods to investigate the rheological impact of arbitrarily shaped nanoparticle additives in shearing interfaces. The rheology of particulate suspensions has a long history beginning with Einstein [1], whose seminal paper on Brownian motion demonstrated that the viscosity of a particulate suspension ($\eta$) is solely a function of particle volume fraction ($\phi$) and the viscosity of the suspending liquid ($\eta_s$) as shown in Eq. 49. While Einstein's relationship has been extensively studied and verified experimentally [2–8], theoretically [9–21], and through computational simulations [22–28], recent studies suggest that the addition of nanoparticles can affect the viscosity of a fluid in ways that do not follow Einstein's model [29–32]. Specifically, Einstein's theory only permits the viscosity of a dilute suspension to *increase* with the addition of particulates, while certain experiments show that the viscosity can be made to *decrease* [29–32, 111]. The inherent complexity of suspensions, especially if the matrix fluid of the suspension is considered to be a non-Newtonian fluid, makes it especially difficult to predict the rheological properties from a purely theoretical approach. Thus, there has been great effort recently to develop computational based models to help investigate the properties of these complex fluids. The work contained in this thesis shows good agreement between computational results and experiment dealing with non-Einstein like viscosity reduction with nanoparticle suspensions.

Modeling of suspension flow is especially difficult because of the time evolution of a rigid body motion requires keeping track of a moving boundary at a fluid-solid interface. Couple that with possible non-Newtonian effects of the base fluid, and the computational model can

become extremely complicated. Simulation and analysis of fluid dynamics problems have generally been performed using traditional methods such as the finite difference method (FDM), the finite volume method (FVM), and the finite element method (FEM). With these techniques, a corresponding Eulerian grid (for FDM and FVM), or Lagrangian mesh (for FEM), or both are required to solve the governing equations. However, FEM cannot resolve problems of extreme distortion and Eulerian-based methods have difficulty treating moving material interfaces, deformable boundaries, free surfaces, etc. Mesh free methods offer an alternative for solving problems of computational fluid dynamics (CFD). The most attractive feature being that there is no need for a mesh to solve the problem and therefore no need to provide *a priori* any information about the relationship of the nodes. Of the various mesh free techniques, Smoothed Particle Hydrodynamics (SPH) is the most widely studied method.

Although there has been much development and progress with SPH, there remains a few questions yet to be fully explored: i) Can SPH be utilized to accurately model pressure distributions in full film lubrication interfaces comparable with routine meshed and analytical techniques? ii) Can SPH successfully capture the non-Einstein like viscosity reduction seen recently with nanoparticle additives? iii) Can SPH handle non-Newtonian flows commonly seen in tribological interfaces? The objective of this work is to investigate these questions by creating a SPH code from the ground up and perform fluid simulations in various tribological interfaces.

*Intellectual Merit:* This research will not only greatly advance important features and understanding of SPH, but concurrently will provide insight into various lubricant behavior yet to be fully studied. While there have been many advances with the SPH method, it

has yet to be applied to the field of tribology. Doing so will have the bimodal effect of improving the SPH method and further extending its fields of application while at the same time glean insight into the anomalous results of the non-Einstein like viscosity reduction seen with nanoparticle additives.

*Broader Impact:* Friction and wear dominate the efficiency, energy consumption, heat generation, and lifetime of various machinery worldwide, from automobiles to wind turbines. Proper lubrication of components in such machinery is vital in ensuring the efficiency of the machine as well as to prevent damage in solid contacts. The performance of the lubricant depends on its composition and its physical and chemical characteristics. From the practical engineering viewpoint, prediction of the lubricating film characteristics is extremely important. For fluid lubrication, viscosity is one of the most important parameters that defines the thickness of a lubricant film, which is essential to know in order to prevent asperity on asperity contact. Nanoparticle additives have been reported to improve the properties and performance of lubricants, but it remains essential to create techniques to predict and understand these rheological improvements in a more thorough and systematic way. The purpose of this work is to utilize computational methods to do precisely that.

In addition to expanding the understanding of the rheology of nanoparticle solutions, this work seeks to make SPH a more accessible tool for tribologists who may not be well versed in computational techniques or programming. While there are many commercial software packages available for meshed techniques such as FEM or FDM, there are very few and close to none for mesh free methods. Therefore, most researchers are left with the time consuming and resource intensive procedure of writing and testing their own code. Since much of the code that was used for this research has already been written and tested, we feel it would be

advantageous to release the code so that engineers could use it for their own work. In this way, tribologists could use this tool to design and evaluate next generation nanolubricants. The source files for the code used in this work can be found in Section 7.3.

The specific aims of this research were are as follows:

**SA1:** Model hydrodynamic lubrication in a pad bearing geometry utilizing the SPH method for modeling lubricant flow. Compare predicted pressure field distribution to both lubrication theory and computational fluid dynamics to validate the SPH method for lubrication interfaces (Section 2).

**SA2:** Become more proficient in modeling more complicated geometries with the SPH method. To that end, a study was conducted in fluid flow between meshing involute gear teeth. For further applicability to industry, particulate motion within the flowfield is also studied (Section 3).

**SA3:** Model nanosheets in a non-Newtonian fluid matrix of mineral oil with the SPH method. Validate the model by calculating the viscosity of the pure mineral oil and comparing that to experimental results (apply correction factors to model if results differ). Using the same model, add nanosheets to the fluid to see if non-Einstein viscosity reduction is seen with nanosheets. Compare numerical viscosity results with experiment. (Section 4).

**SA4:** Investigate the effects that surface tension plays with nanoparticle additives in shearing interfaces. Incorporate surface tension modeling into SPH, perform numerical experiments, and see if non-Einstein viscosity reduction is seen with the addition of

nanosheets. (Section 5).

# 1 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is a meshfree, Langrangian, particle-based method initially created to model astrophysical phenomena [33, 34]. Since its introduction, SPH has been extended to applications in continuum solid and fluid mechanics [35–41]. Unlike meshed techniques that rely on the discretization of space into mesh elements, SPH discretizes the mass of a continuum into a set of discrete particles. These particles not only act as interpolation points for continuum properties (e.g. density, velocity, pressure, etc.), but also carry material properties and exchange energy and momentum according to the continuum, constitutive relations. SPH has been shown to be suitable for modeling moving or deforming boundaries, multiphase fluids, and free surfaces [40].

## 1.1 Equations of Motion

The motion of a continuum, excluding the force of gravity, is governed by the continuity equation

$$\frac{1}{\rho}\frac{\mathrm{D}\rho}{\mathrm{D}t} = -\frac{\partial u^\alpha}{\partial x^\alpha} \tag{1}$$

and by the Cauchy equation of motion

$$\rho\frac{\mathrm{D}u^\alpha}{\mathrm{D}t} = \frac{\partial \sigma^{\alpha\beta}}{\partial x^\beta} \tag{2}$$

where the total time derivative is taken in the moving Lagrangian frame of reference, $u_\alpha$ is the velocity component in the $\alpha$ direction, $\rho$ is the material density, $\sigma$ is the total stress tensor, $t$ is time, $x$ is the position vector, and $\alpha$ and $\beta$ represent indices of the principal directions in the domain with repeated indices implying a summation (Einstein notation).

The total stress tensor $\sigma$ can be further decomposed into the isotropic pressure $P$ and viscous stress $\tau$ as follows:

$$\sigma^{\alpha\beta} = -P\delta^{\alpha\beta} + \tau^{\alpha\beta} \tag{3}$$

When modeling Newtonian fluids, the constitutive relationship between the viscous shear stress and shear strain rate is given by

$$\tau^{\alpha\beta} = \eta\epsilon^{\alpha\beta} \tag{4}$$

where $\eta$ is the dynamic viscosity of the fluid and $\epsilon$ is the shear strain rate.

Modeling fluid flow with SPH, the particle motions are driven by the stress tensor gradient while the pressure is explicitly calculated by the local particle density. Therefore, when modeling incompressible flow, an artificial compressibility has to be introduced to produce the time derivative of pressure. This approach is known in literature as weakly compressible SPH and is the most widely used implementation of SPH. Another technique, known as incompressible SPH, computes the fluid pressure $P$ by implicitly solving the pressure Poisson

equation [42]. While this can increase the size of the time steps in the simulation, it comes at the computational cost of solving an elliptic partial differential equation (PDE) at every time step.

Calculating the pressure term in the momentum equation is a formidable task for simulating incompressible flows, not just for SPH, but for other numerical methods such as finite difference method [43]. Since the actual equation of state for an incompressible fluid would lead to prohibitively small time steps for stability (due to the Courant-Friedrichs-Lewy (CFL) condition [44]), an artificial equation of state is adopted. The artificial equation of state introduces an artificial compressibility. It is based on the fact that every incompressible fluid is theoretically compressible and, therefore, it is feasible to use a quasi-incompressible equation of state to model the incompressible flow. For situations in which free surfaces are not present, a popular equation of state used in literature is

$$P = c^2 \rho \tag{5}$$

where $c$ is the speed of sound, $P$ is the pressure, and $\rho$ is the density.

This equation of state was used by Morris et al. [40] to model low Reynolds number incompressible flows using SPH. By using a scale analysis, Monaghan [39] was able to show that the density variation $\delta^*$ is

$$\delta^* = \frac{\Delta\rho}{\rho_0} = \frac{V_b^2}{c^2} = M^2 \tag{6}$$

where $V_b$ is the bulk fluid velocity and $M$ is the Mach number. If the actual speed of sound is used for $c$ in Eq. 5, not only will it lead to excessively small time steps, but the Mach number will also be very small. This would lead to a density variation that is nearly negligible. Since the pressure is calculated explicitly from the density, a sufficient density variation must occur in the simulation to produce the desired pressure fluctuation. Morris et al. [40] found that the computed pressures were in close agreement with other techniques when $c$ is chosen such that the density varies by at most 3%. He also found that the speed of sound should be comparable with

$$c^2 = \max\left(\frac{V_b^2}{\delta^*}, \frac{\nu V_b}{\delta^* L}, \frac{FL}{\delta^*}\right) \tag{7}$$

where $\nu$ is the kinematic viscosity, $F$ is a body force per unit mass, and $L$ is a characteristic length scale. Choosing $c$ is usually an iterative method in which a low resolution simulation is run to see the actual variation of $P$, after which the value of $c$ can be changed to achieve the desired density fluctuation or pressure distribution.

## 1.2 Kernel and Particle Approximations

There are two key steps in applying the SPH method to the conservation equations. The first step is the *kernel approximation* in which the field functions are represented in integral form. The second step is the *particle approximation* in which the domain is discretized into an arbitrary set of particles.

In the first step, it is first assumed that every function within the governing equations

can be represented in integral form as shown

$$f(x) = \int_{\Omega} f(x')\delta(x - x')\mathrm{d}x' \tag{8}$$

where $x$ is the position vector and $\delta(x - x')$ is the Dirac delta function, given by

$$\delta(x - x') = \begin{cases} 1 & x = x' \\ 0 & x \neq x' \end{cases} \tag{9}$$

The integral representation in Eq. 8 is exact and rigorous. However, by replacing the delta function with a smoothing function $W$, the integral representation can be approximated with second-order accuracy [43, 45–48] as follows:

$$f(x) \approx \int_{\Omega} f(x')W(x - x', \lambda)\mathrm{d}x' \tag{10}$$

where $\lambda$ is the smoothing length describing the domain of the smoothing function.

A number of different smoothing functions have been used in the SPH method. The function is usually chosen to be an even function and must satisfy a number of conditions. One such stipulation is the *unity condition* as follows:

$$\int_{\Omega} W(x - x', \lambda)\mathrm{d}x' = 1 \tag{11}$$

which ensures that the integral of the smoothing function over the support domain is unity. Another condition, known as the *compact condition*, transforms an SPH approximation from a global operation to a local one

$$W(x - x', \lambda) = 0 \text{ when } |x - x'| > \kappa\lambda \tag{12}$$

where $\kappa$ is a constant related to the smoothing function, which defines the influence area of the smoothing function. This condition changes the integration over the entire problem domain into a localized integration over the support domain (influence area) of the smoothing function. The most widely used smoothing function in the SPH literature, and the one chosen for this research, is the cubic spline function (Eq. 13 & Fig.1) [40, 49, 50]. Its popularity is most likely due to the fact that it resembles a Gaussian function while having a narrower support domain (thus leading to less particle interactions and increasing the computational efficiency).

$$W(d, \lambda) = \frac{10}{7\pi} \begin{cases} 1 - 3d^2/2 + 3d^3/4 & 0 \leq d < 1 \\ (2 - d)^3/4 & 1 \leq d < 2 \\ 0 & d \geq 2 \end{cases} \tag{13}$$

The relative distance $d$ between two particles is defined as

$$d = \frac{|x - x'|}{\lambda} \tag{14}$$

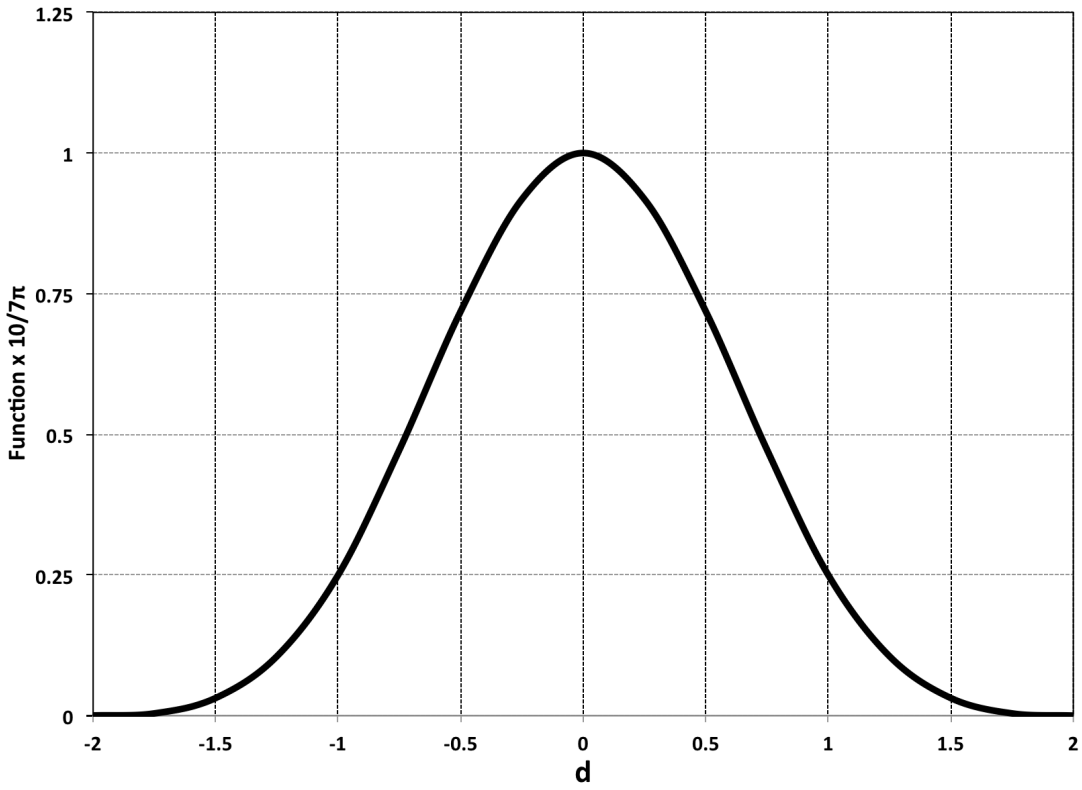


**Figure 1:** The cubic spline kernel.

The next step in the SPH formulation is to perform the *particle approximation*. This step allows us to discretize the continuous integral representations shown in Eq.10 into a summation over particles within a support domain. Figure 2 depicts the support domain surrounding particle $i$.

The support domain is spherical with the radius being the product of a constant $\kappa$ and smoothing length $\lambda$. The smoothing length need not be a constant and can vary both

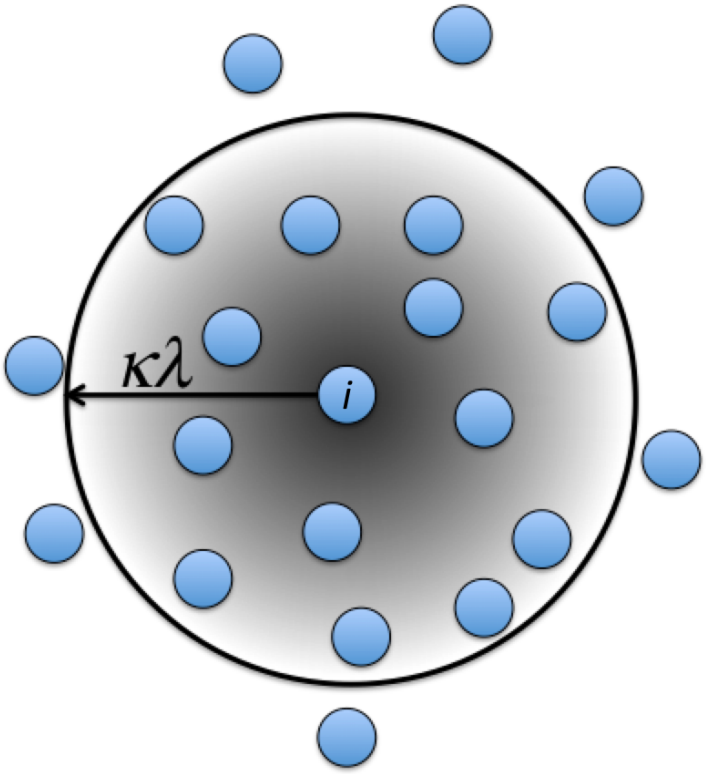


**Figure 2:** Particle approximation for particle $i$. The support domain is spherical with radius $\kappa\lambda$. The color gradient represents the weighting of the smoothing function.

spatially and temporally. Therefore, the support domain size can be inversely proportional to the local density, thus ensuring enough particle interactions. The color gradient represents the weighting of the smoothing function. Note how particles closer to particle $i$ are weighted more heavily than ones near the edge of the support domain and that particles outside of the support domain are not included at all in the calculation. Instead of an integral sign applied to the entire domain $\Omega$, it is replaced with a summation sign over a finite number

of neighboring particles. The infinitesimal volume in Eq. 10, $\mathrm{d}x'$, can also be replaced with the finite mass and density of the neighboring particles as follows:

$$f(x_i) \approx \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(x_j) \cdot W(x_i - x_j, \lambda) \tag{15}$$

where $m_j$ and $\rho_j$ are the mass and density of particle $j$, respectively, and $N$ is the total number of neighboring particles for particle $i$. Similarly, the gradient of $f$ is calculated from

$$\nabla_\alpha f(x_i) \approx \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(x_j) \cdot \nabla_\alpha W(x_i - x_j, \lambda) \tag{16}$$

These last transformations have now converted the continuous integral representations of a function into a discretized weighted summation based on an arbitrary set of particles. It should also be noted that in Eq.16, the spatial gradient on the right hand side has been moved from the field variable to the smoothing function; making calculating derivates relatively trivial (since the smoothing function is usually assumed to just be a piecewise polynomial). This particle approximation is performed at every time step, and hence the use of the particles depends on the current local distribution of particles. Using this procedure, the conservation equations (Eqns. 1 and 2) can be modified from a continuum form into a particle form:

$$\frac{1}{\rho_i} \frac{\mathrm{D}\rho_i}{\mathrm{D}t} = -\sum_{j=1}^{N} \frac{m_j}{\rho_j} u_j^\beta \cdot \frac{\partial W_{ij}}{\partial x_i^\beta} \tag{17}$$

$$\frac{\mathrm{D}u_i^\alpha}{\mathrm{D}t} = \sum_{j=1}^{N} m_j \frac{\sigma_i^{\alpha\beta} + \sigma_j^{\alpha\beta}}{\rho_i \rho_j} \frac{\partial W_{ij}}{\partial x_i^\beta} \tag{18}$$

where $W_{ij}$ is shorthand for $W(x_i - x_j, \lambda)$.

It should be noted that these transformations are not unique. Several other transformations can be derived; however, these are the most prevalent in the SPH literature and the ones used in this research. Equation (18) has been symmetrized with the two stress terms to reduce error arising from particle inconsistency [47, 49].

## 1.3 Density Approximations

There are two approaches for evolving the densities of the SPH particles (the mass of each particle remains constant but the density can change depending on the concentration of surrounding particles). One approach to approximate the density for use in Eq. 18 is known as *summation density*, and simply uses the particle approximation in Eq. 15 by replacing $f(x)$ with $\rho$ resulting in

$$\rho_i = \sum_{j=1}^{N} m_j W(x_i - x_j, \lambda) \tag{19}$$

This states that the density at particle $i$ is simply approximated by the weighted average of the surrounding densities of the particles in that support domain (the unit of the weighting

function W is the inverse of volume). This approximation conserves mass exactly since the integration of the density over the entire problem domain is exactly the total mass of the particles. One issue in using this approach is spurious density approximations at the boundaries of the domain. This occurs because only particles inside the boundary contribute to the average and no contribution comes from the outside. One possible improvement, suggested by Randles and Libersky [51], is to normalize the right hand side of Eq. 19 with the SPH summation of the smoothing function itself

$$\rho_i = \frac{\sum_{j=1}^{N} m_j W(x_i - x_j, \lambda)}{\sum_{j=1}^{N} \left(\frac{m_j}{\rho_j}\right) W(x_i - x_j, \lambda)} \tag{20}$$

This expression helps improve the accuracy of the SPH method near free surfaces as well as material interfaces when the summation is taken only on particles of the same species. Particles located in the interior of the domain can still use Eq. 19 to evolve density.

The second approach is to evolve the density from the continuity equation (Eq. 1). Using this, we can arrive at that following *continuity density* form:

$$\left(\frac{D\rho}{Dt}\right)_i = \sum_{j=1}^{N} m_j(\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W(x_i - x_j, \lambda). \tag{21}$$

The biggest disadvantage with this method for calculating the evolution of the density is that it does not conserve mass exactly (unlike the summation density approach above). However, the continuity density approximation does not suffer from any edge effects. Ad-

ditionally, the summation density approach requires more computational effort because the density must be evaluated before other parameters can be interpolated as well as requiring the calculation of the smoothing kernel itself. In most simulations, the summation density approach is utilized unless there are free surfaces in the problem, in which the continuity approach can be applied only to particles within a smoothing length of $\kappa\lambda$ inside from the free surface.

## 1.4 Boundary Treatment

One of the last hurdles in fully exploiting the SPH method is the complication in dealing with particles near or at a boundary. The first difficulty lies in setting up a boundary in which fluid particles cannot penetrate. One approach to this problem, proposed by Monaghan [39], is to use boundary particles that apply a repulsive force similar to a Lennard-Jones potential of the form

$$f(r) = \begin{cases} D\left[\left(\frac{r_0}{r_{ij}}\right)^{n_1} - \left(\frac{r_0}{r_{ij}}\right)^{n_2}\right]\frac{x_{ij}}{r_{ij}^2} & r_{ij} \leq r_0 \\ 0 & r_{ij} > r_0 \end{cases} \tag{22}$$

where $n_1$ and $n_2$ must satisfy the condition of $n_1 > n_2$ and are usually taken as 12 and 4, respectively. The cutoff distance $r_0$ determines when the repulsive force is applied and is usually taken as the initial particle spacing. $D$ is a problem dependent parameter and should be taken as the square of the largest velocity. It has been shown that this type of treatment for boundaries is very stable and prevents unphysical particle penetration [43].

The repulsive boundary particles, however, are not completely sufficient in resolving the

problems associated with the boundary. Another issue, known as *particle deficiency*, occurs due to the fact that the kernel and particle approximations are truncated near a boundary as shown in Fig. 3.

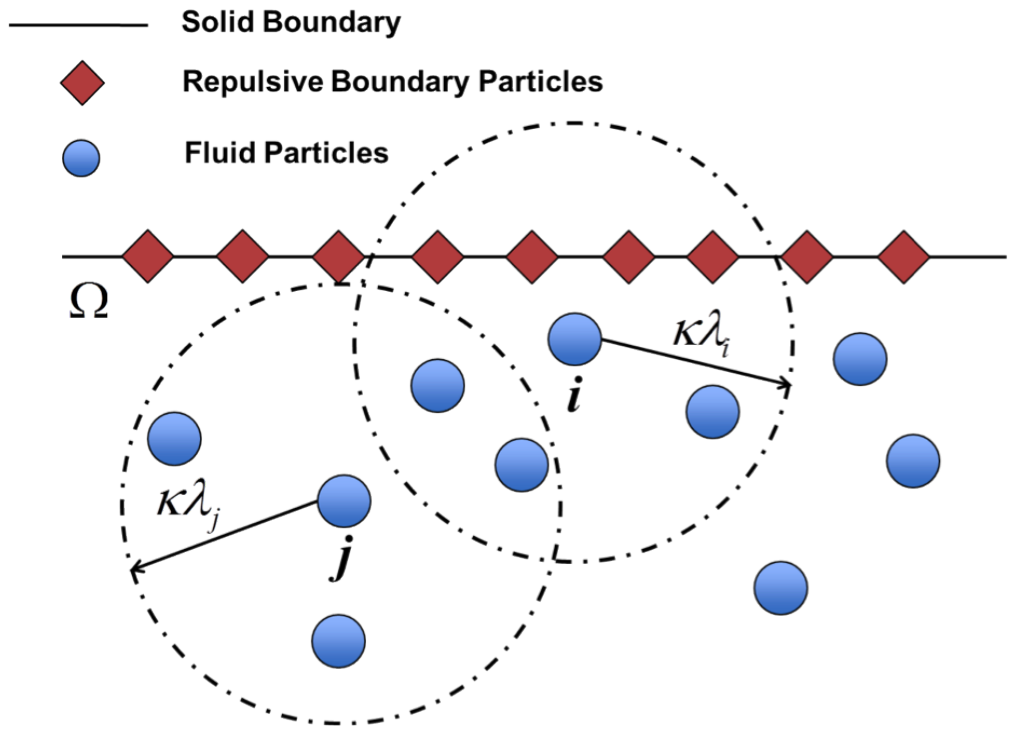


**Figure 3:** Illustration depicting particle deficiency near a boundary for particle $i$. Particle $j$ has a support domain completely contained within the domain, therefore, not experiencing any deficiency.

The one-sided contribution for particle $i$ will lead to incorrect solutions since field variables will be reduced to zero. This particle deficiency occurs not only at boundaries, but also at free surfaces as well as material interfaces. This problem will be addressed on a case by case basis with varying complexity.

## 1.5 Fluid-Structure Interaction Model

For simulating solid structures within the fluid flow, a different set of particles are utilized in the domain. These solid particles do not move in relation to one another, thereby simulating a rigid solid. For particles that are located a distance greater than $\kappa\lambda$ from the interface, only particles of the same species are within the support domain (e.g., particle $i$ in Fig. 4). The most straightforward approach, as suggested by Antoci et al. [35], is to

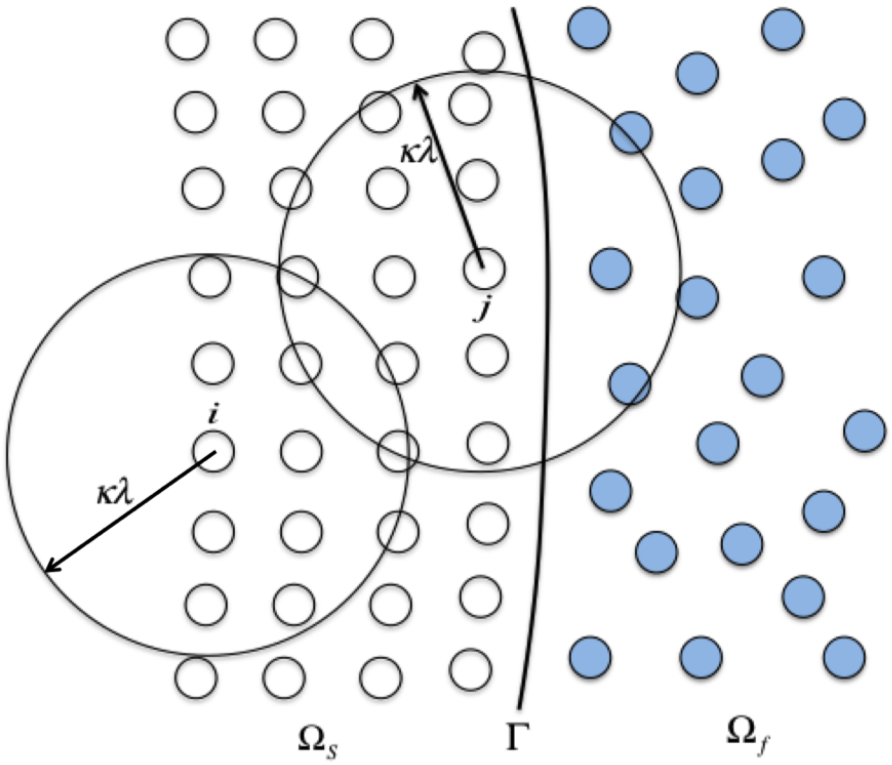


**Figure 4:** Fluid (filled) and solid (unfilled) particles interacting near an interface.

extend the summation in Eq. 18 to all particles, regardless of their nature. This is similar

to introducing coupling conditions between a solid and viscous fluid as follows:

$$u_f^\alpha = u_s^\alpha \tag{23}$$

and

$$\sigma_{ij_s} n_{j_s} = -\sigma_{ij_f} n_{j_f} \tag{24}$$

where the subscripts $f$ and $s$ are for the fluid and solid, respectively. This approach automatically imposes a no-slip condition on the contact interface and counteracts the particle deficiency problem at material interfaces.

While this fluid-structure interaction is convenient, it does not prevent unphysical particle penetrations from occurring. To resolve this, a technique proposed by Monaghan called XSPH [47, 48] is usually utilized (the X is not an acronym and thus stands for nothing). With this technique, the velocity term is somewhat averaged, allowing particles to move in a velocity closer to the average velocity of the neighboring particles

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = u_i - \zeta \sum_j \frac{m_j}{\rho_j}(u_j - u_i) W_{ij} \tag{25}$$

where $\zeta$ is a constant in the range of $0 \leq \zeta \leq 1$. This technique also resolves some instabilities when modeling incompressible flows. In most situations, $\zeta = 0.3$ is usually sufficient in simulating most flows and was used in this research.

## 1.6 Nearest Neighbor Particle Search

With SPH, the smoothing function has a compact support domain, and thus only a finite number of particles are within the support domain of dimension $\kappa\lambda$. These particles within the support domain are then used in the particle approximations given in Sections 1.2 and 1.3. These particles are generally referred to as nearest neighboring particles (NNP) for the particle of interest. The process of finding the nearest particles is commonly referred to as nearest neighboring particle searching (NNPS). As opposed to grid-based numerical methods, where the mesh preserves the position of neighboring grid-cells and is well defined once they are given (unless severe mesh deformation is present in which case numerical accuracy is compromised), the nearest neighboring particles in the SPH method for a given particle can vary with time.

A direct and simple NNPS algorithm is the *all-pair search* approach (as shown in Fig. 5(a)). For a given particle $i$, the all-pair approach calculates the distance $r_{ij}$ from $i$ to each and every particle $j$ $(= 1, 2, \cdots, N)$, where $N$ is the total number of particles in the problem domain. If the distance $r_{ij}$ is smaller than the dimension of the support domain for particle $i$, $\kappa\lambda$, particle $j$ is found belonging to the support domain of particle $i$. Since the smoothing length is often symmetric, particle $i$ is also within the support domain of particle $j$. Therefore, particles $i$ and $j$ are a pair of neighboring particles. This search is then performed for all particles. The all-pair search approach is carried out for particles $i = 1, 2, \cdots, N$, and the searching is performed for all particles $j = 1, 2, \cdots, N$ resulting in a complexity approaching the order $O(N^2)$. Since the NNPS process is necessary at each and every time step (unless more complicated algorithms are implemented in which neighboring

particles are assumed to be constant over a predefined number of time steps), this approach is intolerable for cases with a large number of particles and is rarely utilized unless very few particles are being simulated and/or the problem is only one or two dimensional.

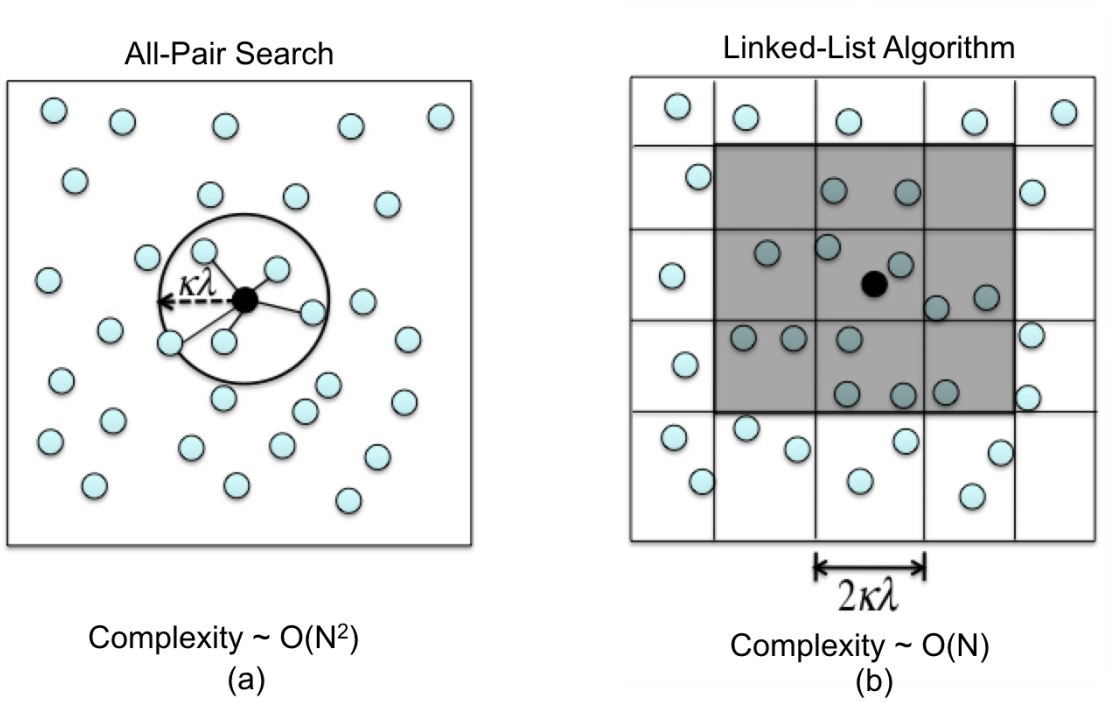


**Figure 5:** (a) All-pair search algorithm for searching for nearest neighboring particles. For each particle, the distance from another particle is compared with the dimension of the support domain to determine if the two particles are indeed neighbors. (b) Linked-list algorithm utilizing a mesh spaced accordingly with the dimension of the support domain.

The *linked-list* search algorithm works well for cases with spatially constant smoothing lengths. A substantial savings in computational time can be achieved by using cells as a bookkeeping device [52] if all the particles are assigned to cells and identified through linked-lists [53–56]. In the implementation of the linked-list algorithm, a temporary mesh is overlaid on the problem domain (Fig. 5(b)). The mesh spacing is selected to match the dimension

of the support domain. Then, for a given particle $i$, its nearest neighboring particles can only be in the same grid cell or the immediately adjoining cells. The linked-list algorithm allows each particle to be assigned to a cell and for all the particles in a cell to be chained together for easy access. If the average number of particles per cell is sufficiently small, the complexity of the linked-list algorithm is of order $O(N)$.

## 1.7 Time Integration

While SPH could in theory utilize an implicit time integration scheme, it would require the inversion of a large sparse matrix [35]. Therefore, to be more computationally efficient, SPH is usually integrated by the standard explicit leap-frog method. This method calculates the positions and velocities of the particles at interleaved time points (Eqns. (26) and (27)) with the position updated at every whole time step and the velocity calculated from the accelerations (based on Eq. 18) at every half time step. Being of second order, the leap-frog method is an improvement over the first order Euler method.

$$x_{i+1} = x_i + u_{i+1/2}\Delta t \tag{26}$$

$$u_{i+1/2} = u_{i-1/2} + a_i\Delta t \tag{27}$$

To ensure stability, the size of the time steps must satisfy the Courant-Friedrichs-Lewy

condition [44]. This condition simply states that the speed at which numerical information travels must be greater than the maximum speed of physical propagation. This requires the time step to be proportional to the smallest particle resolution, namely, the smoothing length $\lambda$ (Eq. 28).

$$\Delta t \leq 0.25 \frac{\lambda}{c} \tag{28}$$

Besides the CFL condition, there are additional constraints that can determine the maximum size of the time step. Monaghan [48] showed that time steps can be dependent on particle accelerations $f_a$,

$$\Delta t \leq 0.25 \min \left( \frac{\lambda}{f_a} \right)^{1/2} \tag{29}$$

and Morris et al. [40] showed how it can depend on viscous diffusion,

$$\Delta t \leq 0.125 \frac{\lambda^2}{\nu} \tag{30}$$

The coefficients in Eqns. 28-30 can be slightly different depending on the kernel being used and the initial particle configuration. Typically, in either high resolution (small $\lambda$) or high viscosity situations, Eq. 30 is the limiting factor on the size of the time steps. The time step ideally as large as possible, and more complicated techniques can have a variable time step

dependent on the current state of the simulation. If the time step violates any of the above equations, the resulting solution to the simulation becomes completely meaningless.

## 1.8 Method Summary

A summary of simulating quasi-incompressible flow using SPH can now be presented. Fluid particles are initially distributed on a lattice filling all space in a specified geometry. They are assigned an initial density, pressure, and velocity. Solid objects are represented by rigid particles, whose field variables do not evolve with time. At each time step, a nearest neighbor particle search is performed to determine interacting particle pairs. Density is evolved according to either Eq. 19, Eq. 20, or Eq. 21. The equation of the state for the fluid is defined by Eq. 5, and the speed of sound $c$ is chosen such that either density fluctuations are at most 3% or to the desired pressure fluctuation. A cubic spline kernel is used with a smoothing length equal to the initial particle spacing. Particle accelerations are then calculated using Eq. 18 and their velocities and positions are updated by Eq. 26 & Eq. 27 respectively. Figure 6 depicts the steps of the SPH methodology in flow chart form.

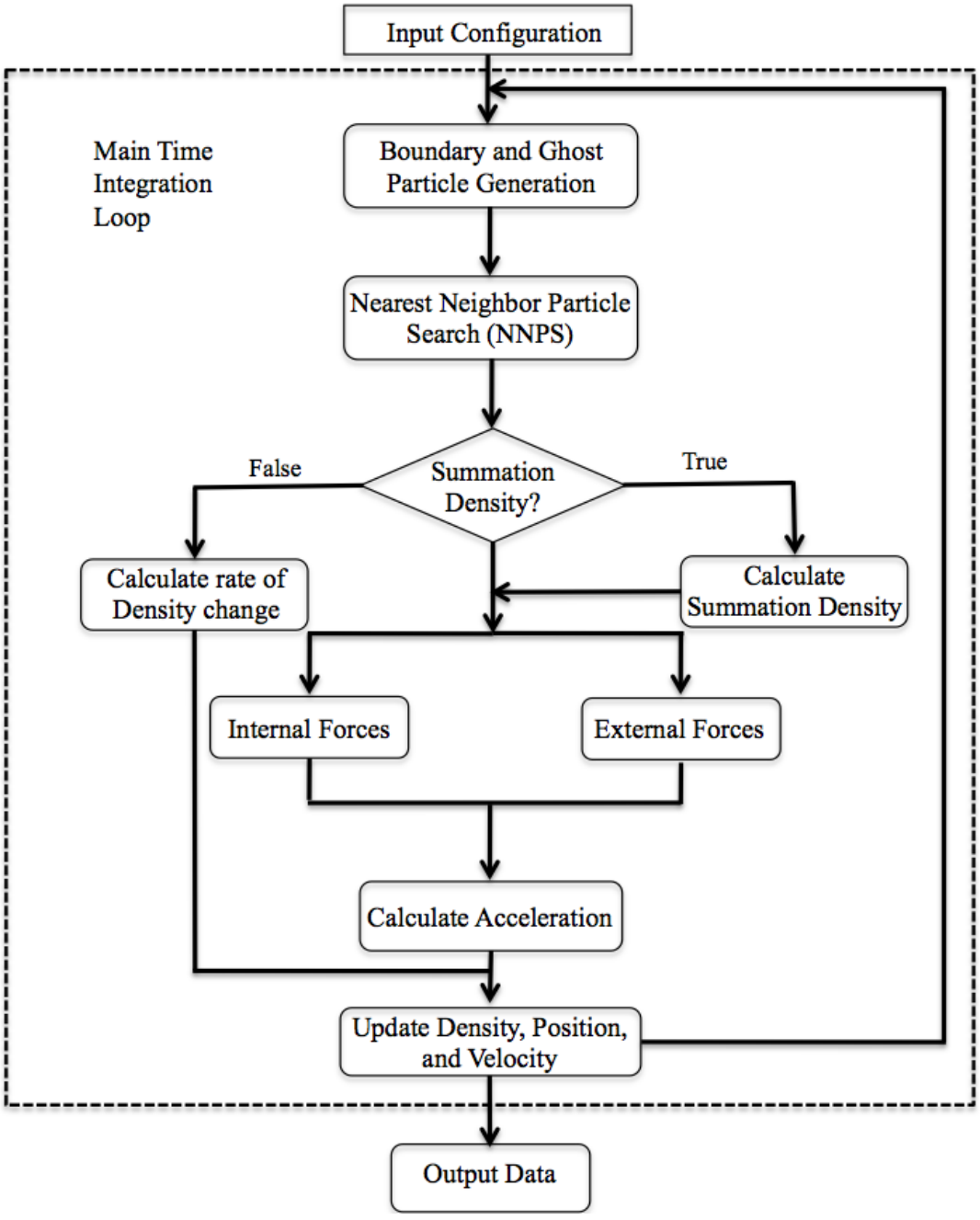


**Figure 6:** Flow chart of SPH methodology. The dashed box represents a single time step.

# 2 Full Film Lubrication Study

An in-house solver was created in order to simulate hydrodynamic lubrication utilizing SPH. In this study, transient hydrodynamic lubrication in a pad bearing geometry was modeled utilizing the SPH method. The results were validated by comparison to computational fluid dynamics (CFD) and an analytical solution provided by lubrication theory. Results for the pressure distribution between SPH and CFD were agreeable while lubrication theory failed to capture any inertial effects of the fluid. Velocity profile comparisons differed slightly between all three methods. However, since smoothed particle methods have been shown to have the advantage of being able to model large deformations, as well as allowing easy definitions of fluid-solid interfaces, they can be useful tools for complex problems in tribology.

## 2.1 Introduction

It is well known that fluid film bearings rely upon the formation of a pressurized hydrodynamic lubricant layer to separate contacting surfaces and minimize wear. The hydrodynamic film itself occurs due to the accumulation of viscous stresses in shearing lubricant, requiring relative sliding motion between surfaces to behave optimally. However, when a sliding surface interface undergoes dynamic speed changes (e.g., during start-up and shut down), the fluid film pressurization effect is compromised and surface contact may occur. Transient behavior of this sort can occur in a variety of interfaces that are subject to transient sliding speeds, such as wind turbine gears and bearings [57], hard disk drive interfaces [58], and artificial knee and hip joints [59]. Because unexpected material wear within such interfaces

can lead to significant repair and/or maintenance costs, it is important to have the ability to predict lubricant behavior under transient conditions.

Several groups have experimentally investigated transient effects in lubricant film formation; this includes works by Glovnea and Spikes [60, 61], Ren et al. [62], and Holmes et al. [63]. A number of additional numerical studies have also been undertaken in order to model transient effects in lubrication, beginning with early works by Herrebrugh [64] and Christensen [65]. More recently, Zhato et al. [66, 67] created a mixed elastohydrodynamic lubrication (EHL) model to predict the lubricant pressure, film thickness, and contact area during start-up. A further study by Popovici et al. [68] used a similar model, combined with the external force from an intertia/spring configuration to investigate startup and oscillatory loading behavior in EHL.

### 2.1.1 Limitations of Classic Lubrication Theory

In modeling studies on transient lubrication, variations of the Reynolds equation [69–73], which relates fluid film thickness $h$ to lubricant pressure $P$, was generally employed. Assuming a one-dimensional sliding, lubricated interface with constant fluid viscosity $\eta$, the Reynolds equation is given as

$$\frac{d}{dx}\left(h^3\frac{dP}{dx}\right) = 6\eta U\frac{dh}{dx} + 12\eta\frac{dh}{dt} \tag{31}$$

where $x$ is a directional coordinate, $t$ is time, and $U$ is the sliding speed. Although the

Reynolds equation has been shown to be applicable for the prediction of a wide variety of lubricant flows [69, 74–77], it was formulated under the assumption of quasi-steady conditions (i.e. neglecting fluid inertial and convective effects), which are important for the study of highly transient lubricant flow domains. The characteristic time scale for the diffusion of fluid information across the width of a lubricant film of thickness $h$ is given by $t_c = h^2/\nu$, where $\nu = \eta/\rho$ is the kinematic viscosity[78]. Assuming an oil film thickness of 10 $\mu m$ and a lubricant viscosity of $\nu$ =4 $cS$ (centistokes) [79], this diffusion scale is $t_c = 0.025$ $ms$. If dynamic events (i.e., changes in sliding speed) occur over a time interval that is of the same order or less than $t_c$, the lubricant will not respond in a quasi-steady-state manner, and solutions based upon the Reynolds equation formulation will be erroneous. It should be noted that attempts have been made to rectify this deficiency in previous works. Pinkus and Sternlicht [80] utilized the method of averaged inertia to account for inertial forces within thin fluid films. This method has been used numerous times such as in the works by Szeri et al. [81] and San Andrẽs and Vance [82] for squeeze film dampers, Elrod et al. [83] for slider bearings, Hashimoto et al. [84] for the short bearing case, and Tichy and Bou-Saïd [85] for hydrodynamic lubrication with impulsive loading. Although results show that this averaging method is better equipped at capturing inertial effects in fluid films as opposed to the classic Reynolds equation, transient startup lubrication effects remain difficult to model.

The objective of this study is to model hydrodynamic lubrication in a pad bearing geometry utilizing the SPH method for modeling the lubricant flow and compare its predicted pressure field to both lubrication theory and computational fluid dynamics predictions. Additionally, the dynamic functionality of SPH was applied to lubrication theory by investigating the transient hydrodynamic pressure within a pad bearing interface during start-up.

## 2.2 Modeling Domain

To simplify the simulation of hydrodynamic lubrication, we decided to model a pad bearing geometry. These types of bearings, consisting of a pad sliding over a surface, are used quite extensively in engineering applications to sustain thrust loads from shafts. Figure 7 depicts the bearing geometry of a single pad. The fluid is initially dragged into a converging region separating two bodies and then exits out from the space. This system automatically equilibrates itself until the induced pressure balances the applied load and the inflow equals the outflow [86]. The film thickness varies along the geometry of the bearing and is represented by $h$.

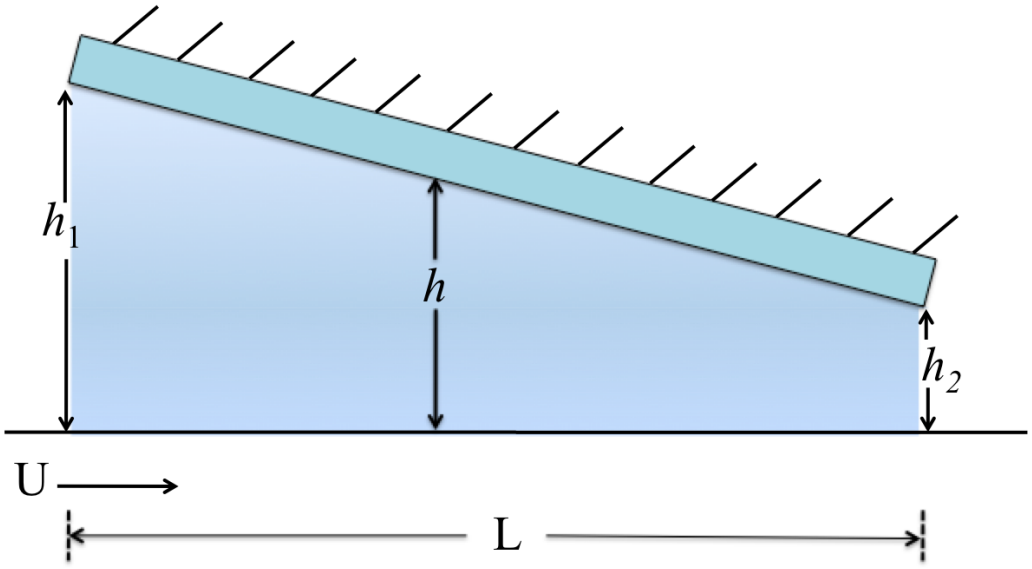


**Figure 7:** Diagram of sliding pad bearing geometry. The pad is fixed while the opposing surface moves to the right at velocity $U$. The bearing width is represented by $L$ while the film thickness is $h$.

## 2.3 SPH Model

For the SPH model, we utilized a sliding wedge geometry submerged in a long fluid channel to simulate the pad bearing geometry. A fluid channel 1.65 cm long and 0.025cm tall was used for the computational domain. The fluid was modeled to have an initial density of $\rho = 1200$ kg/m$^3$ and a dynamic viscosity of $\eta = 0.001$ Pa s. The incompressible fluid is modeled as slightly compressible as discussed in Sec. 1.1 with a speed of sound $c$ of 0.03 m/s and an initial pressure of 0 Pa.

The wedge was modeled to have an inlet height ($h_1$) of 170 $\mu$m and an outlet height ($h_2$) of 70 $\mu$m with a bearing length ($L$) of 1.52 mm. The fluid within the channel wedge was initially static (Fig. 8) in order to allow transient effects to be captured during the start-up of sliding. At an initial time, the wedge was simulated to move laterally with a constant velocity of 0.008 m/s. A convergence study was run (Fig. 9) to determine the appropriate number of fluid particles to model the fluid with. To compromise between numerical accuracy and computational resources, it was deemed acceptable, due to the less than the 3% load error, to model the fluid with 75,000 fluid particles. The simulation is run for 250,000 time steps, with each time step being equal to 10 $\mu$s, for a total simulation time of 2.5 seconds. It was run on a 12-core machine and took 32 h to complete.

No-slip boundary conditions were enforced on the fluid at the boundary and wedge walls. This was done automatically by extending the summation in Eq. (18) to all particles as discussed in Sec. 1.5. Utilizing XSPH also ensured that neighboring particles of different species do no penetrate one another. The simulation was run until the pressure distribution under the wedge reached steady state conditions.

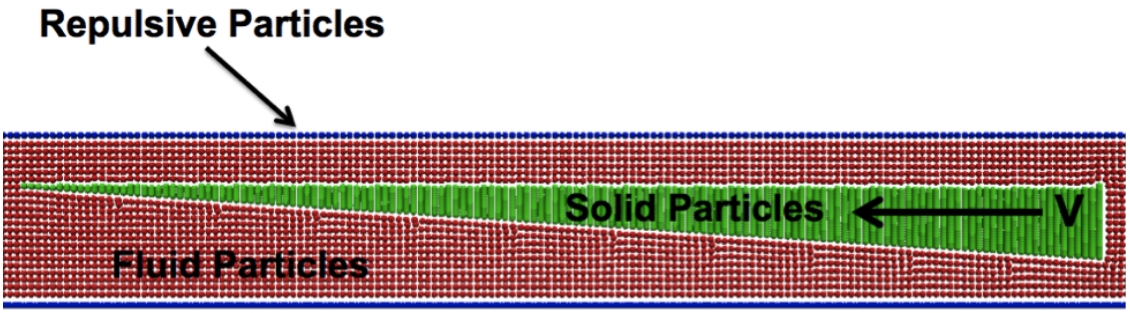


**Figure 8:** SPH simulation of sliding wedge geometry

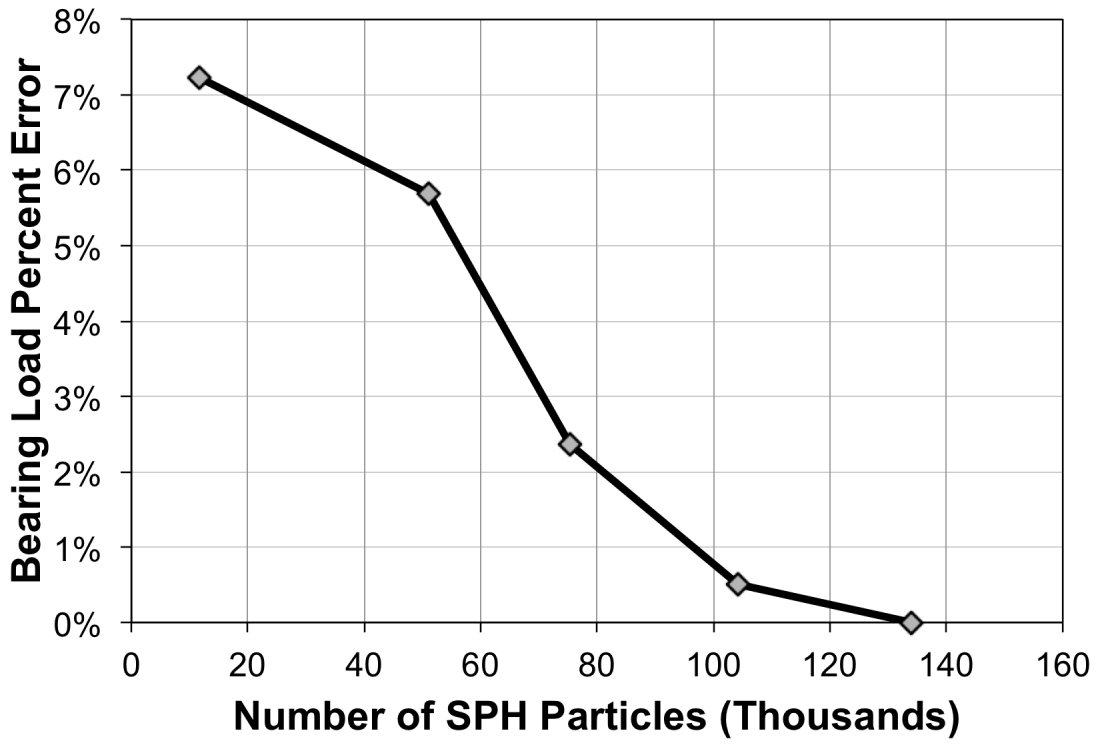


**Figure 9:** SPH convergence study. Relative percentage error of the bearing load at steady state was calculated.

## 2.4 Results and Discussion

The pressure underneath the wedge was averaged across the film thickness and nondimensionalized as follows:

$$P^* = \frac{(P - P_2)(h_2)^2}{U\eta L} \tag{32}$$

where $P$ is the fluid pressure, $h_2$ is the film thickness at the outlet, $P_2$ is the pressure at the outlet, $U$ is the sliding velocity, $\eta$ is the dynamic viscosity of the fluid, and $L$ is the wedge length. Figures 10 and 11 depict the evolution of the nondimensionalized pressure distribution inside the interface. The location of maximum pressure experienced underneath the wedge occurs approximately at $x/L = 0.8$. This location remains relatively unchanged during the transient start-up conditions with only the magnitude varying.

It can be seen that the inlet pressure ($x/L = 0$) is appreciably higher than the outlet pressure. This can be attributed to a "ramming" effect, or inertial pressure rise, caused by the wedge moving within the channel.

It is usually assumed that the inlet pressure to a pad bearing or wedge is either zero or identical to atmospheric pressure [87]. By not accounting for the deceleration or acceleration of the lubricant when it flows around the wedge, it ignores the possibility of an inertial pressure rise. This pressure rise has been verified both experimentally [58], computationally [88], and analytically [89]. The effect of an increased inlet pressure on the bearing is a higher load capacity [90], and thus, predictions produced by classical hydrodynamic theory provides a margin of safety on bearing design.

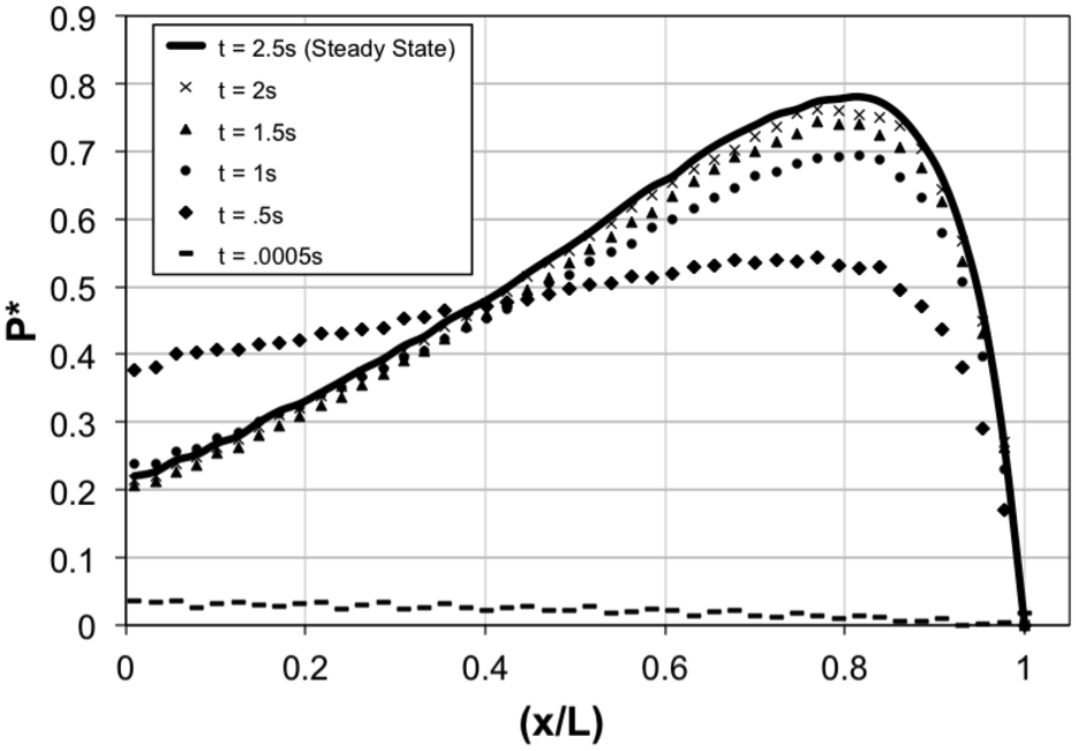


**Figure 10:** Evolution of hydrodynamic pressure distribution with time

The determination of steady state lubrication was analyzed by monitoring the time variation of load carrying capacity $W$ (Fig. 12), calculated via midpoint numerical integration of the hydrodynamic pressure field. The load carrying capacity was nondimensionalized and normalized as follows:

$$W^* = \frac{\left(\frac{W}{U\eta L}\right) - W_0^*}{Max\left(W^*\right)} \tag{33}$$

where ${W^*}_0$ is the initial normalized bearing load (the initial bearing load is nonzero due to the slight compression of the fluid from the repulsive boundary particles). The bearing is

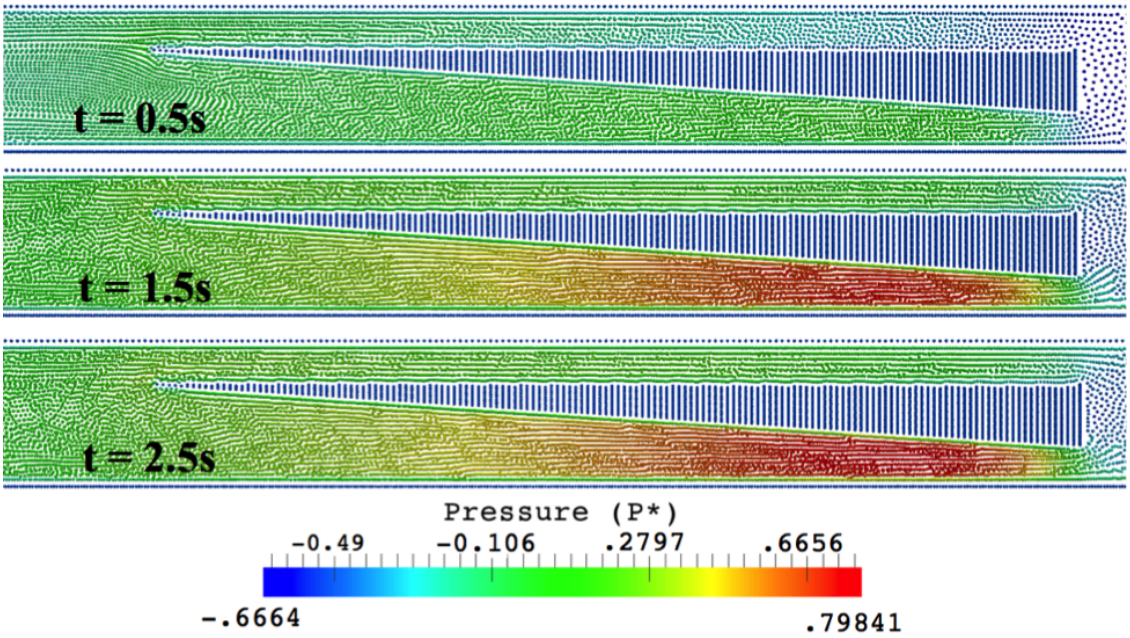


**Figure 11:** Pressure distribution underneath the wedge through time

held stationary for $t = 0.16s$ before moving at a constant sliding velocity.

The pressure distribution at steady state was compared with the analytical solution provided by the Reynolds equation as well as a CFD solution. The CFD solution was provided by a PDE solver solving for the complete Navier-Stokes equations, along with the continuity equation, in two dimensions. For the CFD geometry, a stationary wedge was submerged within a channel using the same dimensions used in the SPH simulation. Figure 13 depicts the meshed geometry of the channel with the submerged stationary pad wedge. The top and bottom walls of the channel were prescribed to move at a constant velocity to the right $U = 8$ mm/s. The horizontal velocity gradient and pressure were set to zero for the inlet and outlet of the channel. The pressure was initialized to be zero. Figure 14 summarizes the boundary conditions for the CFD simulation in schematic form. The pressure was again

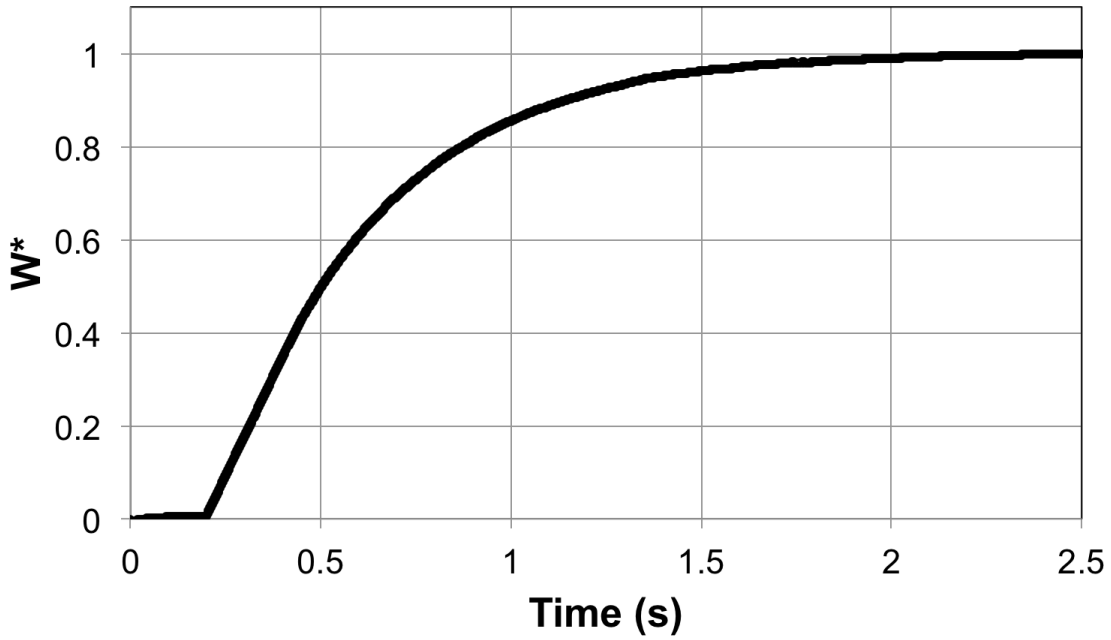


**Figure 12:** Plot of load carrying capacity of the sliding wedge as a function of time. The asymptotically approaching value indicates that the pressure distribution underneath the wedge has reached steady state.

nondimensionalized and Fig. 15 depicts the pressure comparison between the three solutions.

The difference between the inlet pressures, as well as the overprediction of the pressure provided from the SPH simulations and CFD, is a direct result of the ramming effect that occurs due to inertial compression of the lubricant upstream of the moving wedge. This effect has been known in the industry and has provided a margin of safety for designing pad bearings based on classical hydrodynamic theory. It can be seen though that the pressure trend is similar between all predictions.

Figure 16 depicts the horizontal velocity profile underneath the pad bearing at varying

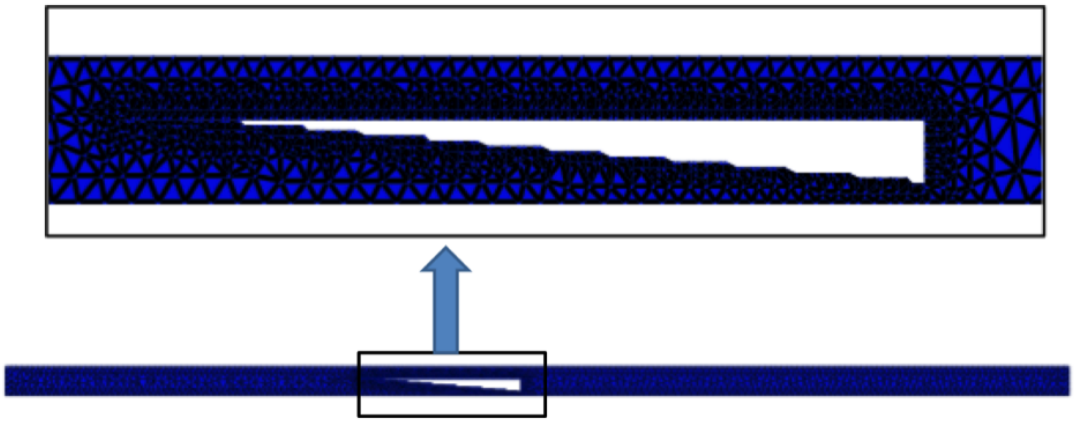

**Figure 13:** Meshing scheme for the CFD model of the sliding pad bearing

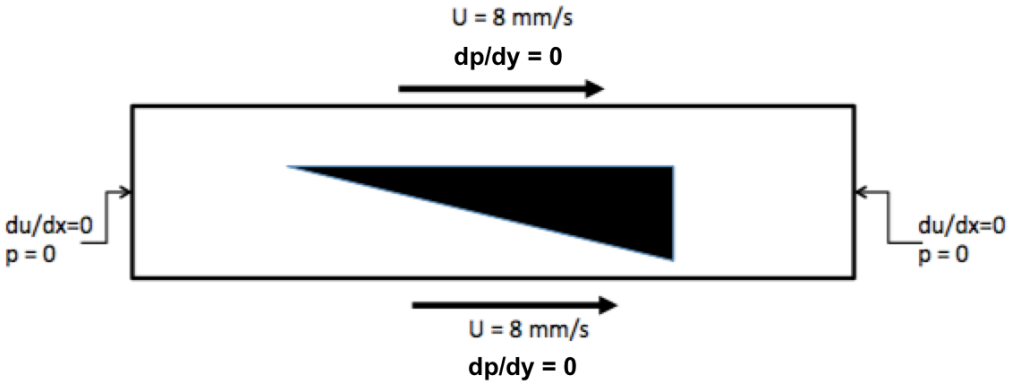


**Figure 14:** Schematic of boundary conditions for CFD model of the sliding pad bearing

downstream locations for the SPH, CFD, and lubrication theory solutions. The SPH solution consistently overpredicts the velocity magnitude at each location under the wedge, while the Reynolds solution falls in-between both the SPH and CFD solutions. One possible

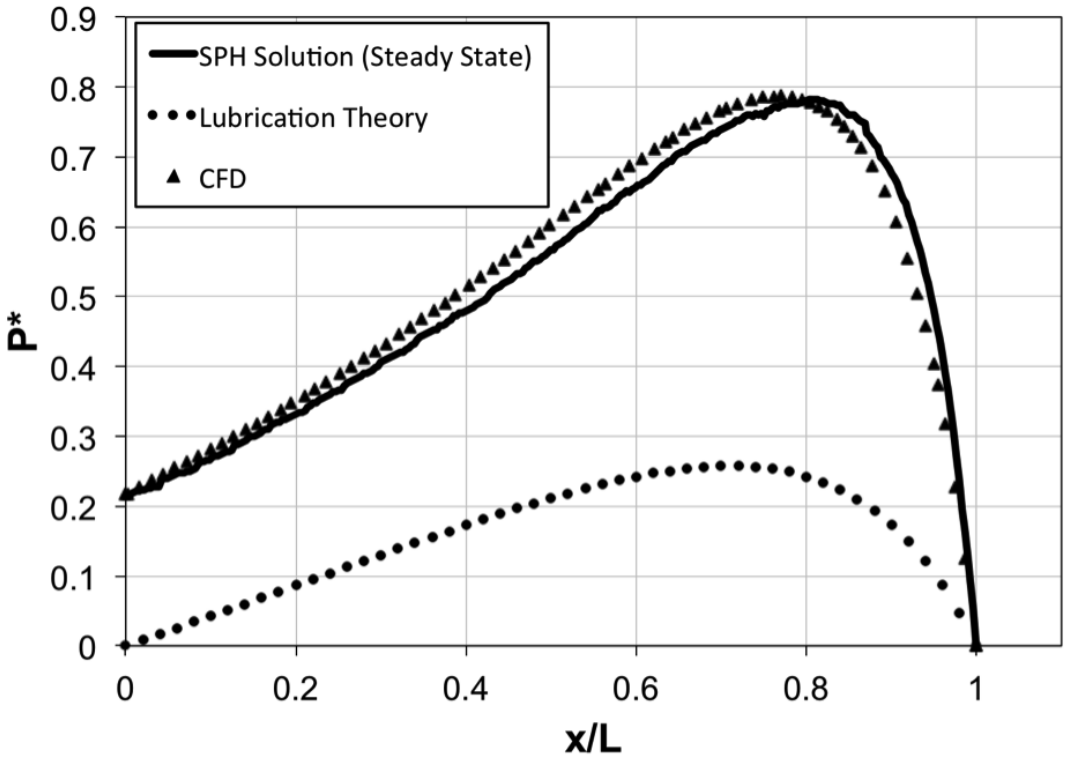


**Figure 15:** Comparison of the pressure distribution between the SPH simulation and analytical solution. The difference in the inlet pressure, as well as the overprediction of the pressure in the SPH & CFD simulations, is a result of the ramming effect of the wedge.

explanation for the slight discrepancy between the SPH and CFD solutions would be the artificial compressibility utilized in the SPH method and reflected in Eq. 5. With the slight reduction in incompressibility for SPH, the solution may be failing to portray the backflow situation seen at the inlet in the CFD solution as well as showing the slight over-expansion at the outlet.

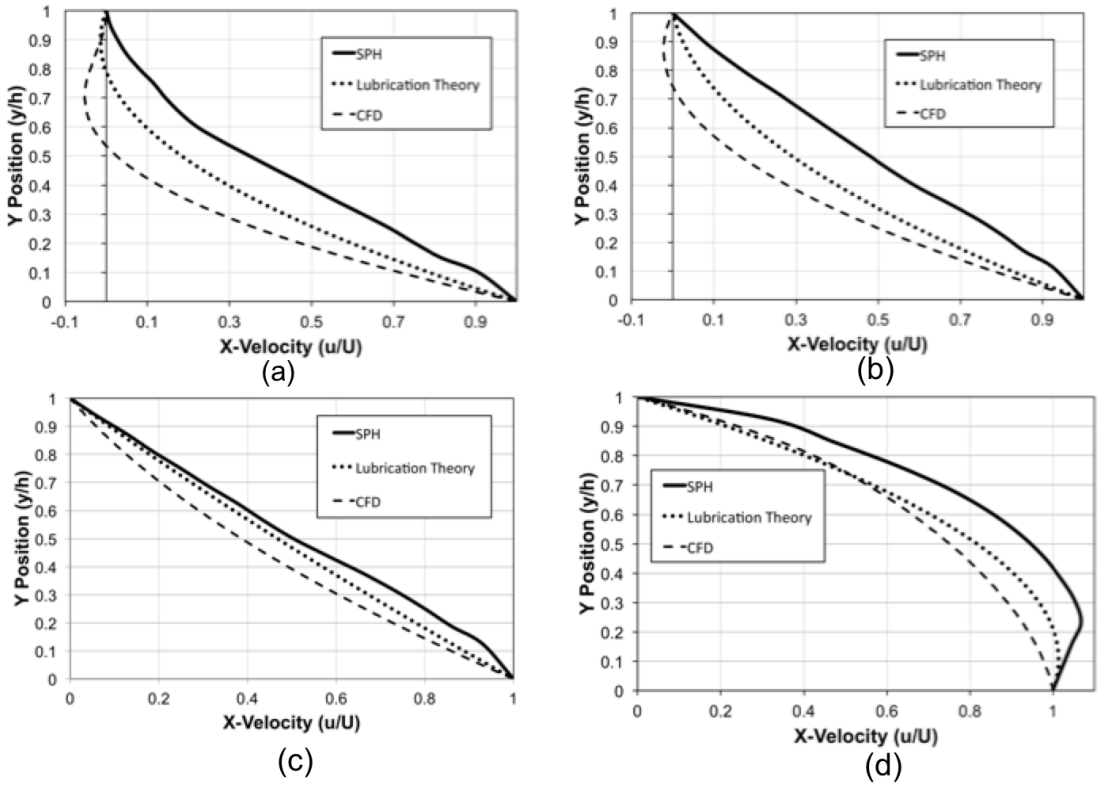


**Figure 16:** Comparison of velocity profile prediction between SPH, lubrication theory, and CFD across the fluid film at (a) $x/L = 0$ (inlet), (b) $x/L = 1/3$, (c) $x/L = 2/3$, and (d) $x/L = 1$ (outlet)

## 2.5 Conclusion

A sliding wedge geometry was created to simulate a pad bearing geometry. In the model, both the solid wedge and fluid particles are discretized by SPH particles. Wall particles, generating a repulsive force similar to a Lennard-Jones potential, were utilized to create a boundary around the entire channel. No-slip boundary conditions were enforced between both the fluid and wedge walls as well as the fluid and boundary walls.

Transient start-up behavior was observed in the pressure distribution underneath the

sliding wedge. The location of the maximum pressure remains relatively unchanged during the initial sliding process; however, the magnitude steadily increases until asymptotically approaching steady state conditions. This thus corresponds to a drecrease in load carrying capacity during start-up conditions and could contribute to an increase in wear events.

Predictions of steady-state lubricant pressure distribution from the SPH model were compared to the analytical solution provided from the Reynolds equation as well as the CFD solution. An inertial pressure rise upstream of the wedge was noted in both the SPH and CFD simulations that was not captured with Reynolds (due to its neglect of inertial forces). This ramming effect caused the SPH and CFD results to overpredict the pressure distribution when compared to the Reynolds solution; however, the trends in the distribution were similar. The results show that realistic results can be obtained of the pressure distribution of the fluid by the SPH model. Furthermore, transient effects are easily obtainable with the method.

Velocity profile comparisons differed slightly between all three methods. The SPH solution consistently over predicts the velocity magnitude at each location under the wedge, while the Reynolds solution falls in-between both the SPH and CFD solutions. With the slight reduction in incompressibility for SPH, the solution may be failing to portray the backflow situation seen at the inlet in the CFD solution as well as showing the slight over expansion at the outlet.

# 3 Gear-Lubrication Study

In order to gain more expertise in utilizing Smoothed Particle Hydrodynamics in more complicated and dynamic fluid-solid interactions, a study was conducted in modeling fluid flow between meshing involute gear teeth. Because smoothed particle methods have been shown to model large deformations with little degradation in numerical accuracy, as well as being able to sufficiently define fluid-solid interfaces, SPH seems well suited in modeling this complex geometry. For further applicability to industry, particulate motion within the flowfield is also studied. Particulate contaminates can be destructive to any tribosystem and therefore any further understanding of their kinematics in a system can be beneficial. The fluid phase flowfield is obtained by solving the Navier-Stokes equations while the particle trajectories are calculated by integrating a drag force equation of motion. This study demonstrates the effect of particle size being the dominant factor in determining the probability of particle entrainment during a gear meshing cycle.

## 3.1 Introduction

Particulate contamination has widely been recognized as a serious tribological issue that not only affects the reliability and performance of mechanisms and machines, but can contribute to downtime and cost of maintenance [91, 92]. Particulates can originate from various sources that can be either internal or external to the tribosystem. Examples of debris particles generated internally include by-products of machining processes (such as metal chips), core sand from castings, paint flakes, rust, weld spatter, and soot [93]. Internal particles can also consist of wear particles generated by various wear processes such as abrasion, adhesion,

erosion, contact fatigue, pitting, and spallation [94, 95]. Particulates that are generated externally often enter systems due to insufficient sealing from the environment, or are introduced due to repair and maintenance procedures (possibly required due to damage inflicted by internally generated particulate contaminants). Regardless of the origin of these particulates, they can be devastating to any tribosystem and has been the leading catalyst for introducing oil cleanliness standards [96–99].

Gears and transmissions are particularly prone to increased wear rates and vibrations due to the effects of debris [100]. Sari *et al.* [101] showed that sand particles introduced into a lubricant caused significant thickness losses in spur gear teeth in just a few operational cycles and further led to poor surface quality and increased friction. Sayles *et al.* [102, 103] experimentally showed that particulate contamination in rolling contacts led to localized surface defects such as dent formation, and that these formations were the cause of a reduction in fatigue life. Thus, even with proper filtration and capture, particulates can still cause residual damage. It is therefore important not only to predict wear caused by particulate contaminants, but also to determine *when* damage will be caused by looking at the likelihood of particle entrainment in elastohydrodynamic (EHL) contacts [104].

While there have been previous studies of particle entrainment related to tribology [72, 92, 105–107], they have generally studied simplified geometries that either allowed exact solutions of the fluid phase flowfield or ones that could easily be determined through finite difference techniques. The purpose of this study is to study particulate trajectories within a meshing spur gear tooth cycle, a problem not easily solved with meshed techniques. Thus, meshless methods such as SPH can be utilized to help procure the solution. The objective is to simulate contaminant particle motion in an involute gear-tooth meshing cycle utilizing

SPH and investigate the effect particle size has on the probability of particle entrainment.

## 3.2 Modeling domain

The motion of a pair of involute gear teeth was simulated over a single mesh cycle. The geartrain system consisted of two gear wheels, positioned side-by-side (Fig. 17). For each gear, the curved geometries of the gear teeth were defined by the parametric equations of an involute curve, given by:

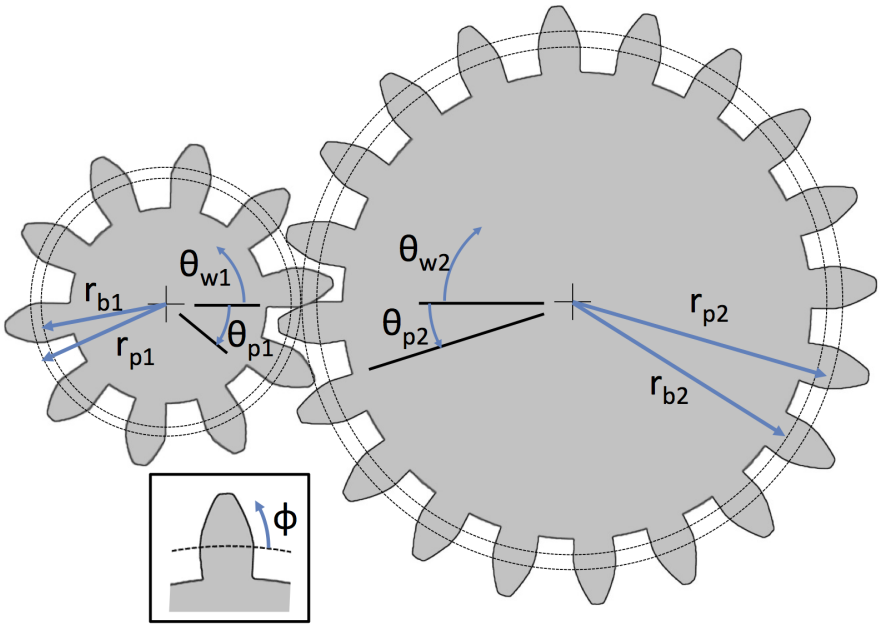


**Figure 17:** Diagram depicting the geometry of two intermeshing gears.

$$x_i(\phi) = r_b(sin\phi - \phi cos\phi) \tag{34}$$

and

$$y_i(\phi) = r_b(cos\phi + \phi sin\phi) \tag{35}$$

where $r_b$ represents the base circle radius of each gear and $\phi$ is the involute angle, which was varied from zero radians to $\phi_{max}$ to form the involute curve of each gear tooth. Rotation of the gear teeth was performed by applying a rotational transformation of the involute curve as follows:

$$x_{ir}(\phi, \theta_r) = x_i cos\theta_r - y_i sin\theta_r \tag{36}$$

and

$$y_{ir}(\phi, \theta_r) = x_i sin\theta_r - y_i cos\theta_r \tag{37}$$

where $x_{ir}$ and $y_{ir}$ are the rotated involute curves of the gear teeth. The rotation angle $\theta_r$ is a sum of the gear pitch angle $\theta_p$ and the angle of rotation of each gear $\theta_w$:

$$\theta_r = \theta_p + \theta_w \tag{38}$$

In setting up the initial geometry, the lefthand gear (Gear 1) was assumed to be centered at the datum of the modeling domain, and therefore the teeth of the righthand gear (Gear 2) were repositioned in reference to the center of the Gear 1 wheel:

$$x_2(\phi, \theta_r) = x_1 cos\theta_r - y_1 sin\theta_r + (r_{p1} + r_{p2}) \tag{39}$$

and

$$y_2(\phi, \theta_r) = x_1 sin\theta_r - y_1 cos\theta_r \tag{40}$$

where $r_{p1}$ and $r_{p2}$ are the pitch radii of Gears 1 and 2, respectively.

The contact point coordinates $(x_c, y_c)$ were determined by numerical calculation of the intersection of the gear tooth involute curves. The meshing gear tooth domain was then referenced with respect to the contact point by subtracting the contact point coordinates from the involute curves of both teeth. As a result, the equations defining the tooth positions on Gears 1 and 2 were as follows:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \left( \begin{bmatrix} sin\phi & -\phi cos\phi \\ cos\phi & \phi sin\phi \end{bmatrix} \times \begin{bmatrix} r_{b1} \\ r_{b1} \end{bmatrix} \right) \times \begin{bmatrix} cos\theta_{r1} & -sin\theta_{r1} \\ sin\theta_{r1} & -cos\theta_{r1} \end{bmatrix} - \begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{41}$$

**Table 1:** Modeling parameters for geartrain domain

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| $r_{b1}$ | 7.0 cm | $r_{b2}$ | 14 cm |
| $r_{p1}$ | 7.5 cm | $r_{p2}$ | 15 cm |
| $\theta_{p1}$ | 0.628 rad | $\theta_{p2}$ | 0.314 rad |
| $\phi_{max1}$ | 0.794 rad | $\phi_{max2}$ | 0.609 rad |

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \left( \begin{bmatrix} sin\phi & -\phi cos\phi \\ cos\phi & \phi sin\phi \end{bmatrix} \times \begin{bmatrix} r_{b2} \\ r_{b2} \end{bmatrix} \right) \times \begin{bmatrix} cos\theta_{r2} & -sin\theta_{r2} \\ sin\theta_{r2} & -cos\theta_{r2} \end{bmatrix} + \begin{bmatrix} (r_{p1} + r_{p2} \\ 0 \end{bmatrix} - \begin{bmatrix} x_c \\ y_c \end{bmatrix} \quad (42)$$

The gear train parameters are shown in Table 1. A single mesh cycle was simulated by varying the gear wheel rotation angle $\theta_w$ between zero and the pitch angle $\theta_p$ over small time increments. Figure 18 depicts the resulting gear tooth motion along a mesh cycle. As a result of the geometric manipulation in Eqn. 41 & 42, the reference frame for the domain was considered a moving reference frame that traveled at the same speed and direction as the contact point.

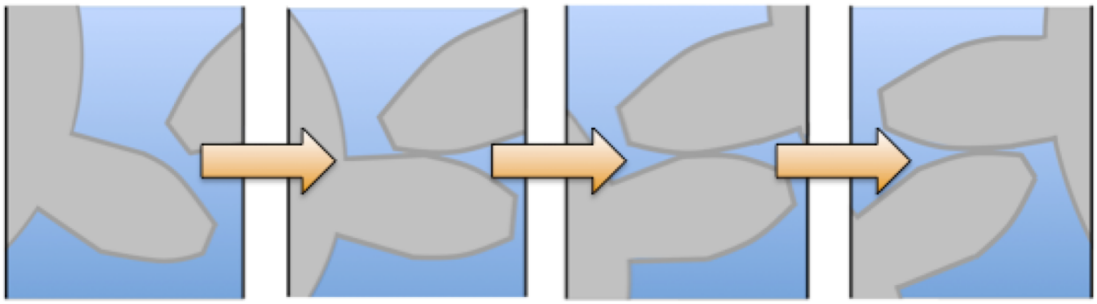

**Figure 18:** Diagram of a pair of involute gear teeth during a single meshing cycle. The gears are immersed within the fluid.

## 3.3 Contaminant Particulate Dynamics

In this study, it was assumed that the presence of the particulates in the fluid would have a negligible effect on the flow field of the particulate-free fluid. This assumption was based on the premise that the particulates being studied were orders of magnitude smaller than the characteristic dimension of the computational domain, and the concentration of particulates was dilute. Thus, the fluid phase flowfield could be calculated independently of the particle motion.

After a solution of the flowfield was calculated with the SPH method (see Section 1), the motion of the particulates could be determined. The governing equations for the particle motion are given as follows:

$$\frac{dx_p}{dt} = u_p \tag{43}$$

$$\frac{dy_p}{dt} = v_p \tag{44}$$

$$m_p \frac{du_p}{dt} = f_{d,x} \tag{45}$$

$$m_p \frac{dv_p}{dt} = f_{d,y} \tag{46}$$

where $x_p$, $y_p$, $u_p$, and $v_p$ are the horizontal and vertical location, and horizontal and vertical velocity of the particle, respectively, and $f_{d,x}$ and $f_{d,y}$ are the drag forces acting on the particle in the x- and y- directions. Eqns. 45 & 46 show the only force considered acting on a particulate was a drag force. While other forces could in theory be acting on a suspended particle such as Saffman lift [108], Magnus [109], and Brownian forces, these were all assumed to be negligible compared to the drag force.

Assuming spherical particulates, the drag forces for Stokes flow over each particle is given as follows:

$$f_{d,x} = 6\pi\mu a_p \left[u\left(x_p, y_p, t\right) - u_p\right] \tag{47}$$

$$f_{d,y} = 6\pi\mu a_p \left[v\left(x_p, y_p, t\right) - v_p\right] \tag{48}$$

where $a_p$ is the radius of each particulate. Since the velocity field of the particulate-free

fluid is nonuniform and transient, it is evident that the drag force on the particulate varies both temporally and spatially during the simulation.

## 3.4 SPH Model

A pair of involute gear teeth, completely submerged within a fluid, was used to simulate the conditions within a conventional transmission. The computational domain (Fig. 19) used was 5 cm wide and 13.2 cm tall with virtual particles around the perimeter applying a repulsive force. The fluid surrounding the gear teeth was discretized into 6,200 fluid SPH particles. The fluid was modeled to have a density of $\rho = 1000\ kg/m^3$ and a dynamic viscosity of $\eta = 0.001\ Pa \cdot s$.

The profile of each gear tooth follows the conventional involute curve geometry commonly used for spur gears. The surface of the gear teeth was modeled with rigid particles as discussed in Section 1.5. The pinion (the lower tooth) was prescribed with a pitch diameter of 15 cm while the gear (the upper tooth) had a pitch diameter of 30 cm. The fluid was initially static in order to allow transient effects to be captured during the start-up of the gear motion. At start-up, the pinion was rotated at 5 rad/s while the gear rotated at 2.5 rad/s. The simulation was run for 10,000 time steps, with each time step being equal to 100 $\mu$s, for a total simulation time of 1 second.

## 3.5 Results and Discussion

Before calculating the trajectory of contaminant particulates, the fluid phase flowfield has to be solved for the meshing cycle. Figure 20 depicts both the motion of the solid gear

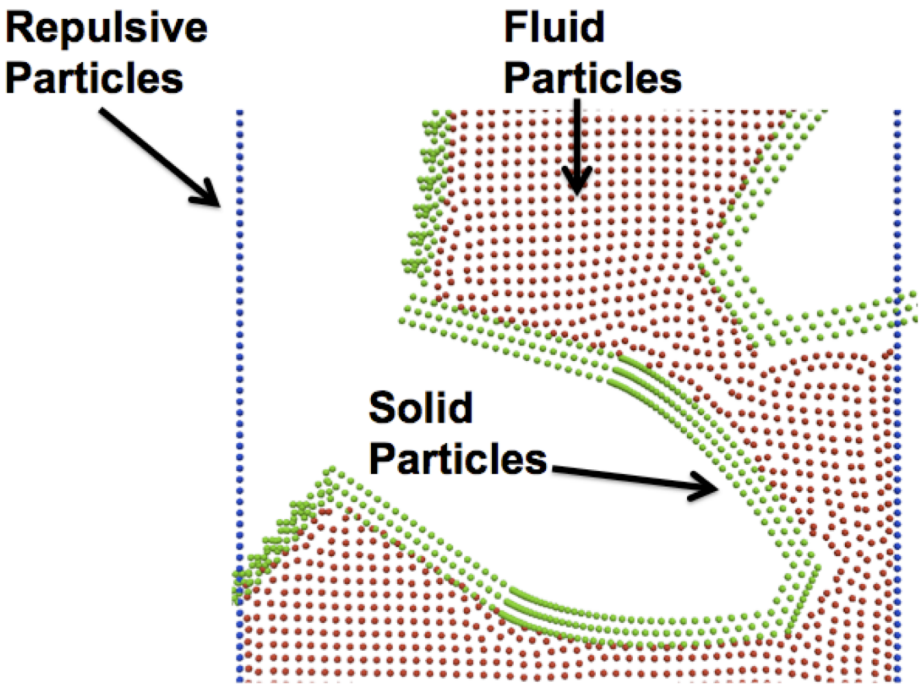


**Figure 19:** SPH Simulation of rotating gear teeth submerged in a fluid. Blue particles represent the repulsive virtual particles, red particles are the fluid SPH particles, and green particles represent the rigid solid particles.

particles as well as the motion of the fluid SPH particles while Fig. 21 shows the velocity flowfield.

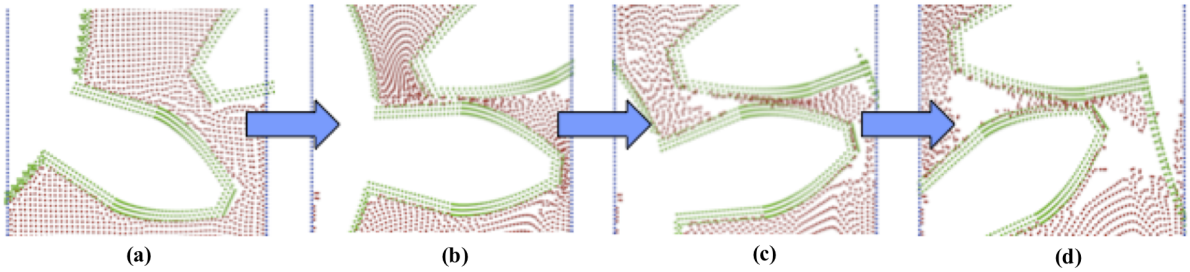


**Figure 20:** Motion of gear teeth and SPH fluid particles during single mesh cycle.

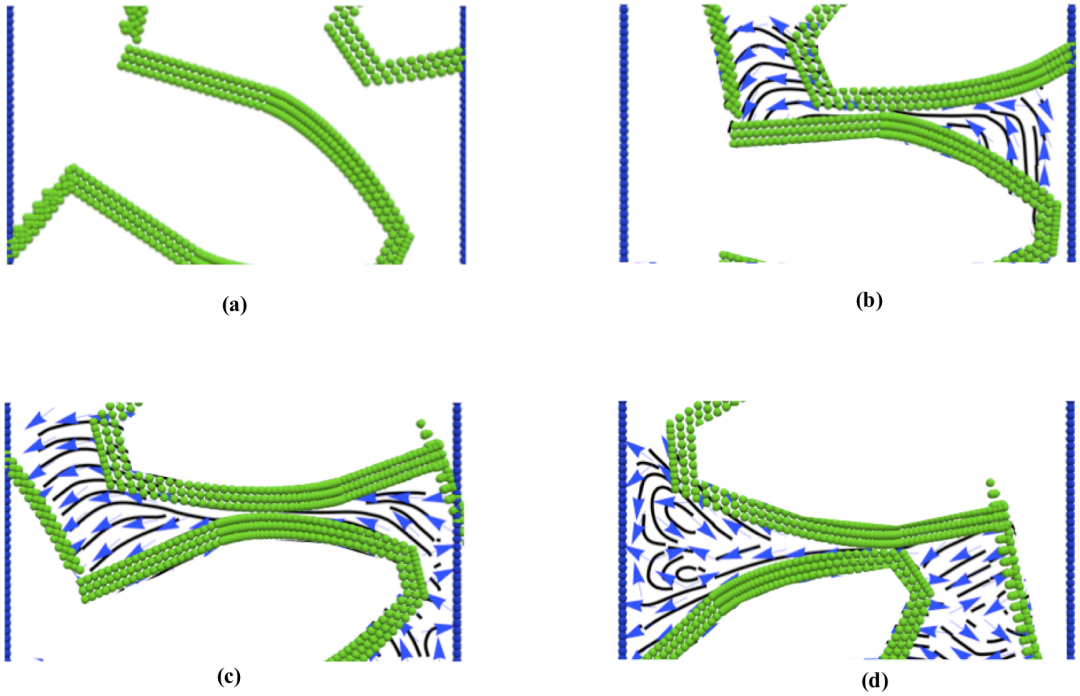


**Figure 21:** Fluid velocity flowfield represented by streamlines and velocity vectors.

It can be seen, especially in Figs. 20(c) and 20(d), that some penetration was experienced between the fluid SPH particles and the solid gear surfaces. Varying the constant $\zeta$ in Eqn. 25 can mitigate nonphysical penetration, although the results were satisfactory since the particles did not penetrate any areas of interest.

It should also be noted that voids were observed in the fluid domain, where SPH particles are not present. One can assume that these areas correspond with cavitation effects experienced during a mesh cycle, which has been shown to occur between meshing gear teeth in oil-lubricated gear boxes transmitting torque (Hunt et al. [110]). Near the end of the cycle, both teeth move away from each other with sufficient velocity to produce these voids.

Once the fluid phase flowfield was determined, the particulate trajectories were then

calculated. As noted in Section 3.3, it was assumed that the particulates had a negligible effect on the flowfield of the particulate-free fluid. Thus, the same flowfield solutions, as depicted in Fig. 21 and Fig. 22, was used for the calculation of all particulates based on the governing equations of motion (Eqns. 43 - 48).

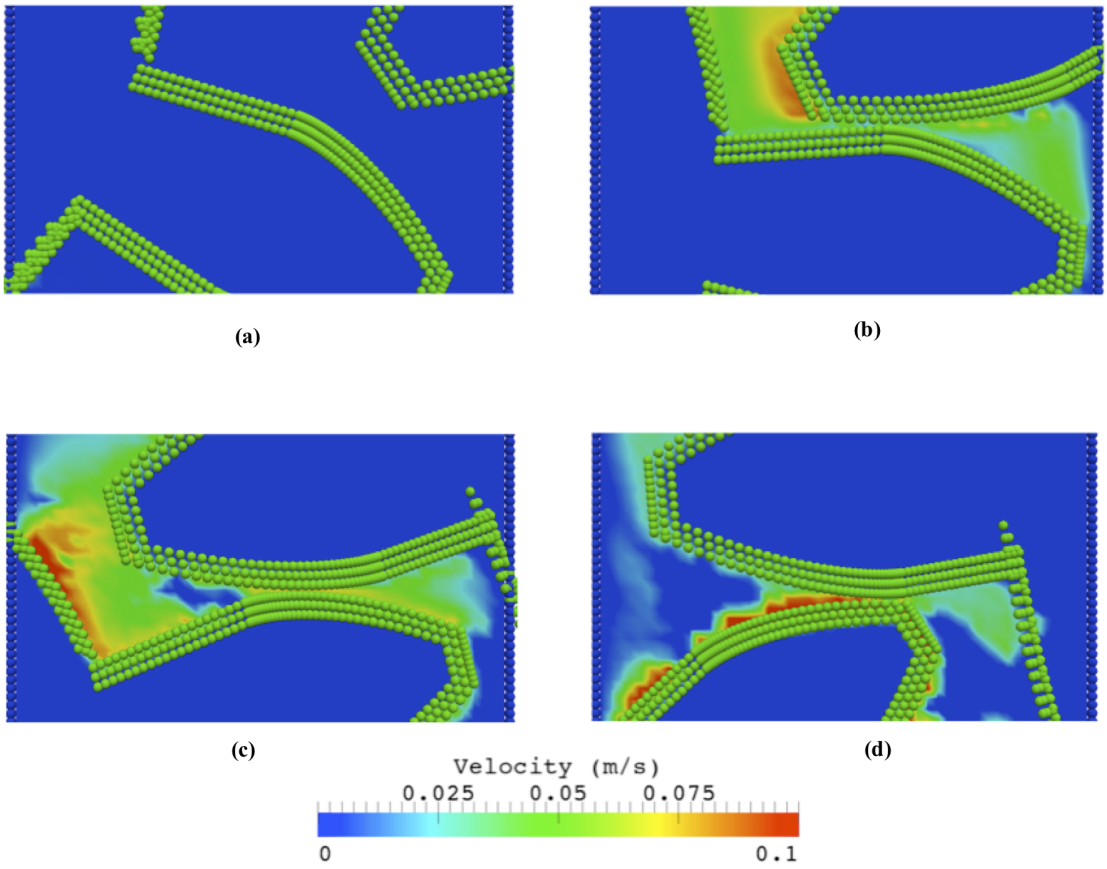


**Figure 22:** Contour plot of velocity magnitude

A total of 214 spherical particulates were introduced into the gear geometry (Fig. 23). They were placed between the leading edge of the pinion tooth, and trailing edge of the gear tooth. The trajectory of the particulates was solely dependent on the flowfield; thus they

did not interact or collide with one another. Therefore, each simulation represented parametrically varying the location of a single particulate and calculated all possible trajectories simultaneously.

Eleven simulations were run with the particulate radius parametrically varied from $a_p = 0.02\mu m$ to $a_p = 1\mu m$. A particulate was considered entrained if at any point during the simulation it satisfied being within a smoothing length $\lambda$ of the stationary contact point while also having any solid particles from the lower pinion tooth as well as the upper gear tooth within its support domain. A total running count of entrained particulates was then calculated for each simulation and is depicted in Fig. 24.

A critical length parameter was calculated to help interpret the data presented in Fig. 24. This critical length was defined as the furthest distance, from the initial particulate location, an entrained particulate was from the stationary contact point. The critical length parameter thus describes the demarcation distance a solid particulate has to be within the contact point, in the initial configuration, to become entrained. This is depicted in Fig. 25.

Thus, it can be easily deduced that with an increase in particulate radius, the percentage of entrained contaminants is greatly reduced. From Fig. 25, it can be seen one explanation of this phenomenon is the linear decrease in the critical length parameter with increasing radius. In essence, larger particulates must initially be closer to the stationary contact point in order for them to become entrained.

Such results have been seen before in literature such as in Nikas [104] study of particle entrainment in EHL point contacts. Because of the reduced size and inertia of the smaller particulates, they can more readily follow the stream lines of the fluid flow, whereas the larger particulates have a more difficult time becoming entrained due to their size. While

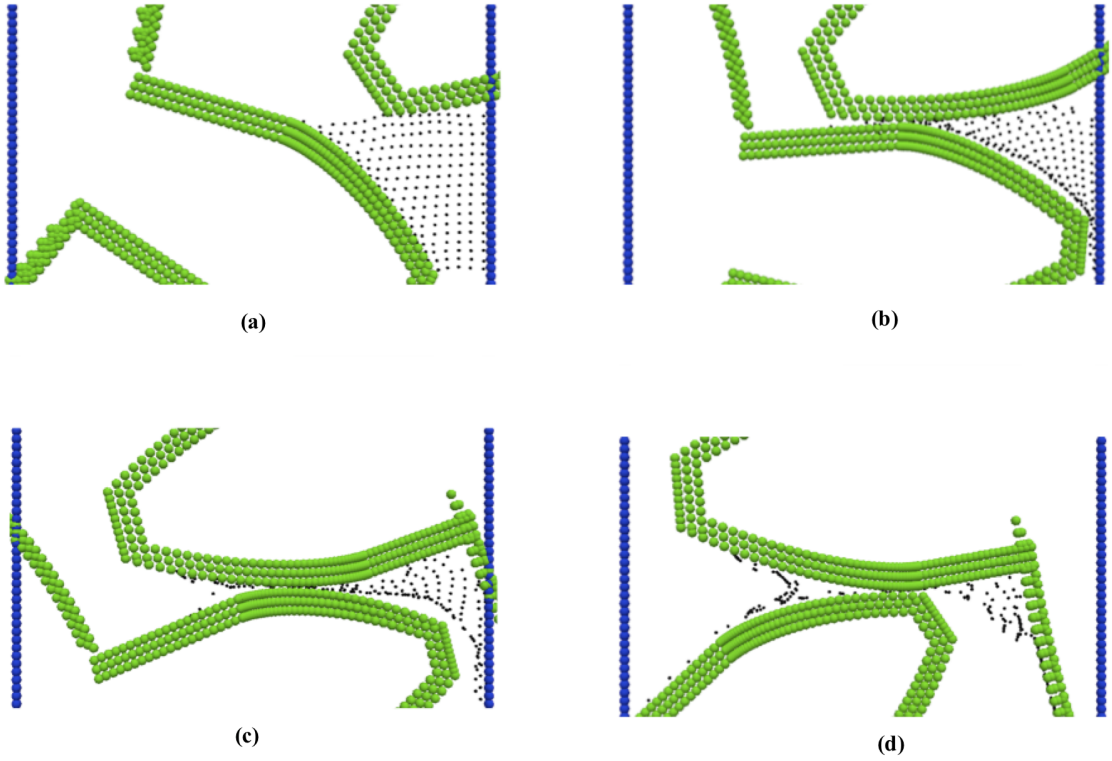


**Figure 23:** Particulate motion during a single mesh cycle.

nothing can be said about the damage being inflicted by these smaller particulates, these results suggest they experience greater interaction with gear surfaces.

## 3.6 Conclusion

A pair of involute gear teeth, completely submerged within a fluid, was used to simulate the conditions within a conventional transmission. In the model, both the solid gear surfaces, as well as the fluid, were discretized into SPH particles. Wall particles, generating a repulsive force similar to a Lennard-Jones potential, were used to contain the computational domain.

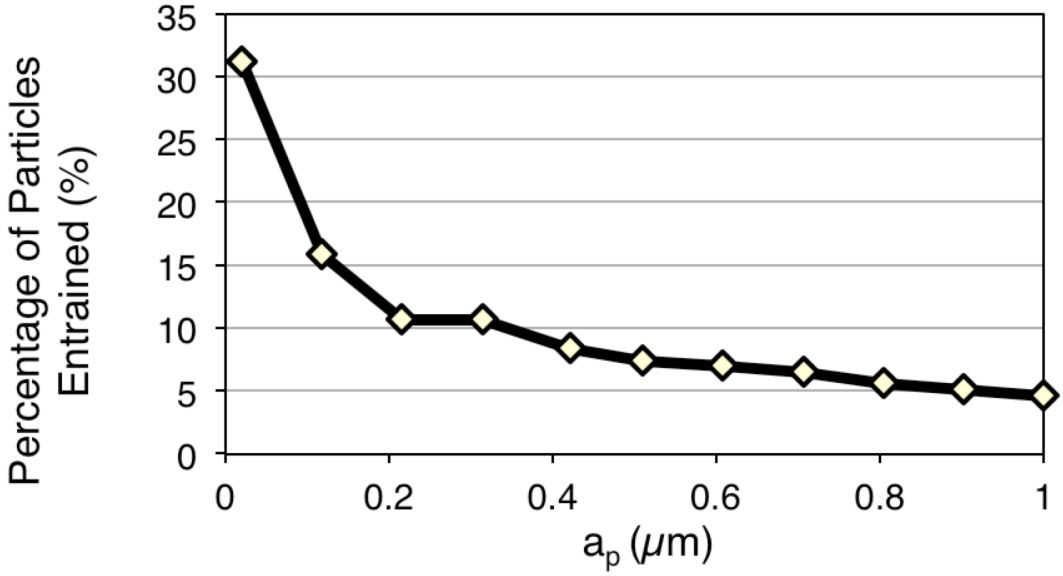


**Figure 24:** Percentage of particulates entrained during meshing cycle for different sized particulates.

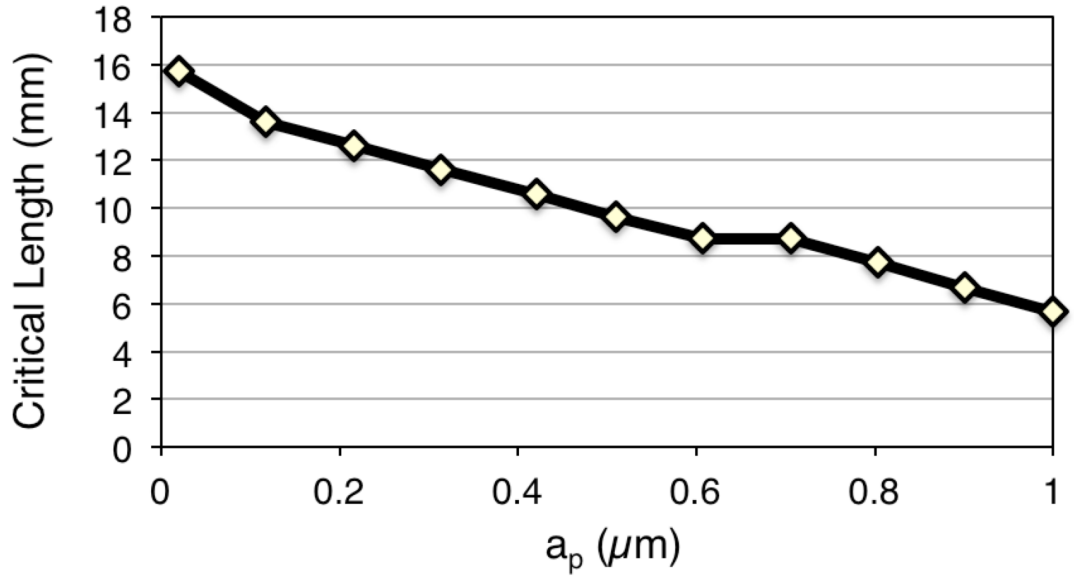


**Figure 25:** Critical length parameter versus particulate radius.

No-slip boundary conditions were enforced between both the gear surfaces and the fluid as well as the boundary walls and the fluid. Contaminant particulates were modeled as spherical particles whose motion was dictated by the solution of the fluid phase flowfield. Drag forces were the only forces assumed to be acting on the particulates.

Particulate trajectories, and entrainment percentages, were calculated for eleven different particulate radii varying from $0.02\mu m$ to $1\mu m$. Results showed a significant drop in particle entrainment as the particulate radii was increased. Calculation of a critical length parameter showed that larger particulates need not initially be closer to the contact point of the gears to become entrained. These results can be attributed to the smaller particulates being able to more readily follow the fluid streamlines due to their reduced size and inertia when compared to the larger particulates.

# 4 Nanocomposite Lubrication Study

The rheological properties of colloidal suspensions has garnered increased interest in a wide array of disciplines including paper manufacturing, oil production and transportation, bioengineering, food processing, and pharmaceuticals. Nanoparticle additives have been shown to improve the mechanical and transport phenomena of various liquids; however little has been done to try and explain the rheological modifications provided from such modifications from a theoretical standpoint. It has been shown [111] that a non-Einstein-like reduction of viscosity of mineral oil is observed with the utilization of yttrium oxide ($Y_2O_3$) nanosheet (NS) additives that do not follow Einstein's relationship as shown in Eq. 49 (where $\eta$ is the viscosity of the total suspension, $\eta_s$ is the viscosity of the base fluid, and $\phi$ is the volume fraction). Experimental results, coupled with generalized SPH simulations, provide insight into the mechanism behind this reduction of fluid shear stress. Two-dimensional nanoparticle additives markedly improve the lubricious properties of the mineral oil, ultimately reducing the friction, and providing a novel way in designing and understanding of the next generation of lubricants.

$$\eta/\eta_s = 1 + (5/2)\phi \tag{49}$$

## 4.1 Introduction

Friction and wear dominate the efficiency, energy consumption, heat generation, and lifetime of machinery. Various additives have been reported to improve properties and performance of lubricants [112, 113]. The function of additives includes deposit control, film-forming, anti-wear, corrosion resistance, and viscosity modification. For fluid lubrication, viscosity is one of the most important parameters that define the thickness of a lubricant film and its shear stress [114]. Viscosity is the measure of the resistance of a fluid under shear. It is expected that additives would affect the shear stress and fluid drag leading to the change in viscosity [115, 116].

Recently developed nanomaterial-based additives are promising in enhancing lubricating efficiency [117, 118]. The two-dimensional (2D) nanocrystals have been studied extensively as solid lubricants [119, 120]. The characteristic of 2D nanostructured materials is their layered structures, with covalent bonds binding each atomic layer together. In between those layers, van der Waals interactions are present. The 2D nanomaterials can also be used as additives in liquid lubricants [121–123]. A 2D nanostructured fluid additive, yttrium oxide ($Y_2O_3$) nanosheet (NS), has recently been discovered to improve the global planarization in chemical-mechanical polishing (CMP) of copper wafers [124]. It has been hypothesized that the 2D NS additive is able to reduce the friction via modifying a lubricant's fluid dynamics via a mechanism that is different from previously reported additives.

In order to investigate the fluidic modification due to the NS additives, computational simulations were performed in which a non-Newtonian fluid is modeled utilizing SPH with the addition of rigid body inclusions [125]. Coupling experimental rheological results with

the computational modeling addresses the origin of the enhanced lubricating performance via viscosity modification. Such findings will shed new light in research in 2D nanostructured particles and their fluidic behavior. The 2D NS-like particles provide an alternative option in developing innovative additives to optimize the dynamic behavior of a liquid lubricant.

## 4.2 Experimental Results

He et al. utilized hydrothermal synthesized $Y_2O_3$ NS as an additive in a base lubricant (mineral oil) to characterize its viscosity enhancement [111]. Detailed results about their synthesis are reported elsewhere [126]. Figure 26 shows a transmission electron microscopy (TEM) image of $Y_2O_3$ NS ($316^{\pm}$49nm side and $16^{\pm}$1nm thick). The effective lubricants consisted of the mineral oil with the NS additives with varying concentrations (1 wt.%, 0.5 wt.%, and 0.1 wt.%). The nanosheets were dispersed in the mineral oil via ultrasonication for 15 min before the measurements. The coefficient of friction was evaluated using a tribometer with pin-on-disk configuration. It consisted of a rotating disk (glass slide) and a fixed E52100 steel ball (6.35mm diameter). The lubricant of 100 $\mu$l was used and the rotational radius was set at 3 mm. The viscosity was measured using an AR-G2 rheometer, varying the shear rate from $10s^{-1}$ to $18740s^{-1}$. A stainless steel parallel spindle (25 mm) rotated, while the lower Peltier plate was stationary. The gap (200 $\mu$m) between parallel plates was filled with the lubricant liquid, and the temperature was maintained at 25 $^oC$.

According to Reynolds' theory, once a continued lubricant film is formed between two bodies in relative motion, a hydrodynamic pressure is built up to separate the two surfaces [127]. The viscosity of the fluid under pressure is a critical parameter that determines the

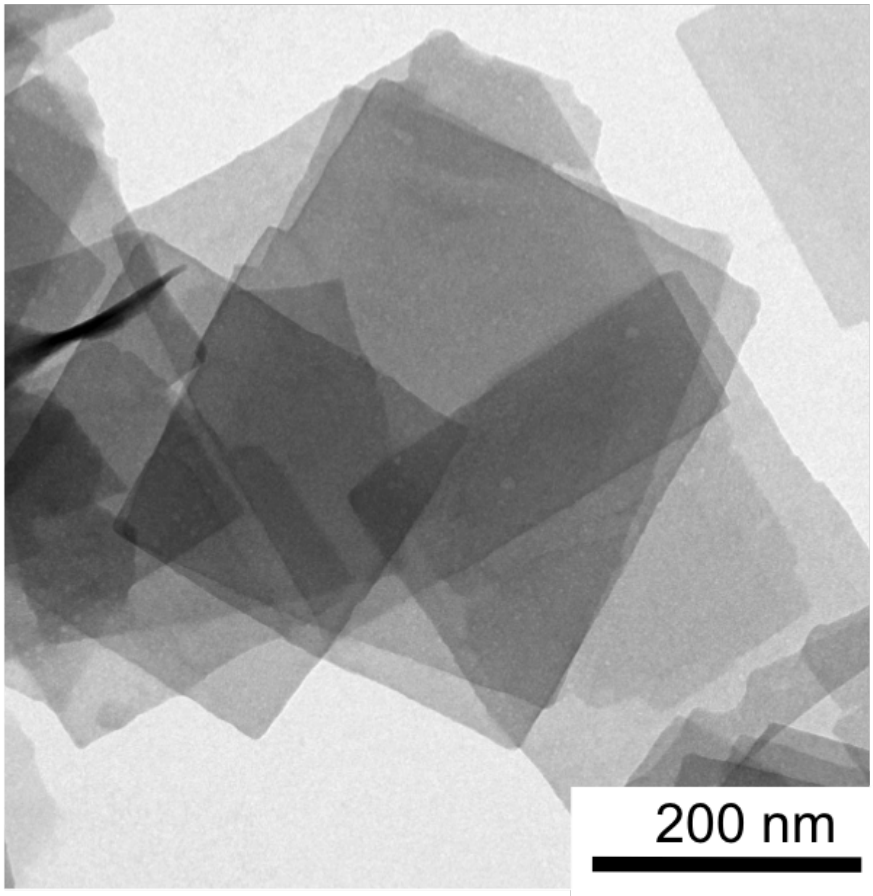


**Figure 26:** TEM Image of $Y_2O_3$ NS.

thickness and performance of a lubricant film. As shown in Fig. 27(a), a low viscosity was obtained once $Y_2O_3$ NS additives were added into the mineral oil. By decreasing the concentration of the $Y_2O_3$ NS, the viscosity was reduced further. At the concentration of 0.1 wt.%, $Y_2O_3$, the NS could reduce the viscosity as much as $\sim 5\%$. The reduction of both the viscosity and coefficient of friction by the $Y_2O_3$ NS additives implied that they had the capability of improving lubrication via modification of the lubricants' rheological properties.

The viscosity reduction with increasing shear rate indicated the shear thinning character-

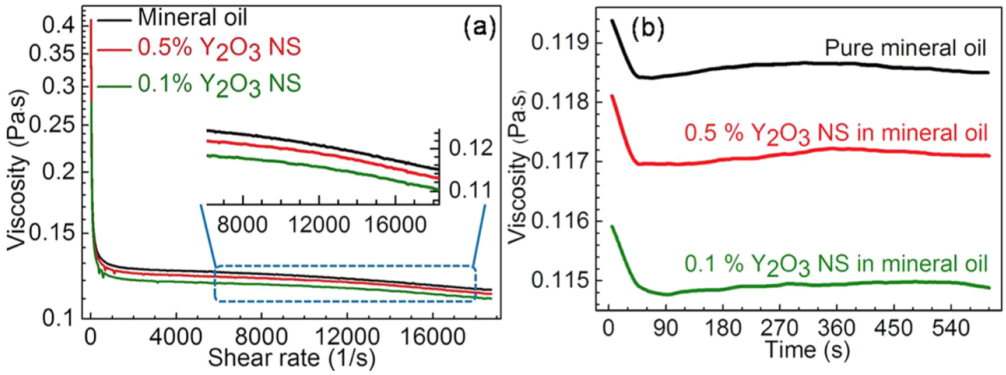


**Figure 27:** (a) Variation of viscosity with shear rate in mineral oil (top black plot), and with addition of 0.5 wt.% (middle red plot) and 0.1 wt.% (bottom green plot) $Y_2O_3$ NS additives. (b) Reduction in viscosity of mineral oil (top black plot) in the presence of $Y_2O_3$ NS with concentrations of 0.5 wt% (middle red plot) and 0.1 wt.% (bottom green plot) under a constant shear rate $(10,000s^{-1})$.

istic of the lubricant [128]. To further understand this phenomenon, a thixotropic study was conducted to investigate the shear thinning properties of mineral oil lubricants with $Y_2O_3$ NS additives. The results are shown in Fig. 27(b). By applying a constant shear rate, the fluid structure was deconstructed initially that led to the quick drop of viscosity at the beginning ($< 60s$). As mineral oil is composed of long-chain alkane molecules, the physical interactions between them enabled the deconstructed microstructure to rebuild continuously. The shear process that broke the molecular structures competed with that which rebuilt the molecular bonding. The dynamic balance between them resulted in a relatively stable viscosity value in time in the later stages ($> 60s$). From Fig. 27(b), the $Y_2O_3$ NS additives were also found to reduce the viscosity in the thixotropic study. The 0.1 wt.% $Y_2O_3$ NS reduced the viscosity more than that of 0.5 wt.% $Y_2O_3$ NS under the constant shear rate.

## 4.3 SPH Model

Modeling of suspension flow is especially difficult because of the time evolution of a rigid body motion requires keeping track of a moving boundary at a fluid-solid interface. Couple that with possible non-Newtonian effects of the base fluid, and the computational model can become extremely complicated. Still, many advances have been made in trying to predict flow properties when the matrix fluid is Newtonian including traditional computational fluid dynamics (CFD), Stokesian dynamics [129], and dissipative particle dynamics (DPD) [130–132]. For this research, a numerical model similar to one presented by Martys et al. [133] was used in which modeling of a non-Newtonian fluid is done by characterizing the fluid using SPH with a spatially dependent viscosity. This model, which builds upon previous work of Monaghan [134] and Español and Revenga[135], allows the viscosity to be dependent on local field variables such as shear rate or temperature whose value can vary spatially and temporally. Coupling experimental rheological results, namely the relationship between shear rate and viscosity, the computational model then has the much more attainable goal of just having to determine local shear rates which can then be used to determine a local viscosity based on empirical data. This local viscosity is then used in the general Navier-Stokes equations to provide the overall flowfield solution of the non-Newtonian fluid matrix.

For this research, simulation results were compared with experimental data for nanoparticles of $Y_2O_3$ diffused in a non-Newtonian fluid matrix of mineral oil. The model is first validated by calculating the viscosity of pure mineral oil and comparing that to experimental results. Using the same model, rigid nanosheets were added to the fluid to see their effects on the total viscosity. These numerical results are also compared to experiment.

In order to model a fluid with spatially variable viscosity, the SPH methodology presented in Section 1 has to be modified slightly. To represent the gradient of a field variable utilizing the SPH method, it is beneficial to slightly change the representation of the weighting function (both for ease in notation as well as continuity with literature). The weight function $W$ is chosen such that the number density, $n_i$, can be written as

$$n_i = \sum_{q=1}^{N} W(x_p - x_q, \lambda) \tag{50}$$

It can be shown that the gradient of $f$ at a point labeled $p$ can be written as

$$\nabla f_p = \nabla f(x_p) = \frac{1}{n_p} \sum_{q=1}^{N} (f_q - f_p)(\mathbf{x}_p - \mathbf{x}_q) F(|\mathbf{x}_p - \mathbf{x}_q|) \tag{51}$$

where $\nabla W(r) = -\mathbf{r} F(\mathbf{r})$ and $r$ is the distance between two particles.

The motion of a particle is a direct result of an effective interparticle force that is present between neighboring fluid particles due to the discretized version of an integral representation of the Navier-Stokes equations. Consider the Lagrangian form of the continuity equation and the general Navier-Stokes equations [136]

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot \mathbf{v}), \tag{52}$$

and

$$\rho \frac{Dv_i}{Dt} = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_k} \left\{ \mu \left( \frac{\partial v_i}{\partial x_k} + \frac{\partial v_k}{\partial x_i} - \frac{2}{3} \delta_{ik} \left[ \nabla \cdot \mathbf{v} \right] \right) \right\} + \frac{\partial}{\partial x_i} \left( \zeta \left[ \nabla \cdot \mathbf{v} \right] \right), \tag{53}$$

where $\rho$ is the fluid density, $P$ is pressure, v is velocity, and $\mu$ and $\zeta$ are the shear and bulk viscosities, respectively. Note that in these equations, the bulk and shear viscosities cannot be moved outside the gradient operator because we are taking them to be spatially dependent field variables. However, in the limit that the viscosities can be taken constant in Eqn. 53, the equation reduces to the more familiar Navier-Stokes equation with constant viscosity.

Given the relationship in Eqn. 51, we can formulate a discrete representation in SPH form of the continuity equation as follows [134]:

$$\left(\frac{D\rho}{Dt}\right)_p = \sum_{q=1}^{N} m_q F(|\mathbf{x}_p - \mathbf{x}_q|)(\mathbf{x}_p - \mathbf{x}_q) \cdot (\mathbf{v}_p - \mathbf{v}_q) \tag{54}$$

For constructing an integral representation of Eqn. 53, see Martys et al. [133] and Zhu et al. [137] to see the integral transformation used to obtain the following result (note that $i$ and $j$ represent directions while $p$ and $q$ represent actual SPH particles):

$$\rho \frac{Dv_i}{Dt} = -\frac{\partial P}{\partial x_i} + A_i + B_i \tag{55}$$

where $A_i$ is defined as,

$$A_i = \nabla \cdot (\mu \partial_i \mathbf{v}) + \nabla \cdot (\mu \nabla v_i) + \partial_i (\mu \nabla \cdot \mathbf{v}) \tag{56}$$

and $B_i$ is,

$$B_i = \frac{\partial}{\partial x_i} \left( \left( \zeta - \frac{5}{3}\mu \right) \nabla \cdot \mathbf{v} \right). \tag{57}$$

The SPH representation of Eqn. 56, defining $A_i$, is as follows:

$$A_p^i = 5\sum_{q=1}^{N} \frac{F(|\mathbf{x}_p - \mathbf{x}_q|)}{\rho_q} \times \left[\frac{(\mu_p + \mu_q)(\mathbf{x}_p - \mathbf{x}_q)^i \cdot (\mathbf{v}_p - \mathbf{v}_q)}{(\mathbf{r}_p - \mathbf{r}_q)^2}\right]. \tag{58}$$

The SPH representation of Eqn. 57 can be done by first utilizing the chain rule of the derivative and taking $a = \zeta - (5/3)\mu$ to clarify notation. One then obtains the following:

$$B_i = \frac{\partial}{\partial x_i}(a\nabla \cdot \mathbf{v}) = (\partial_i a)\nabla \cdot \mathbf{v} + a\partial_i \nabla \cdot \mathbf{v}. \tag{59}$$

Using SPH transformations of the derivatives and gradients [134] in Eqn. 59, one obtains the discretized version as follows:

$$(B_i^a)_p = (\partial_i a)_p = \frac{1}{\rho_p}\sum_{q=1}^{N} m_q F(|\mathbf{x}_p - \mathbf{x}_q|)(\mathbf{r}_p - \mathbf{r}_q)_i(a_p - a_q), \tag{60}$$

$$(B_i^b)_p = (\nabla \cdot \mathbf{v})_p = \frac{1}{\rho_p}\sum_{q=1}^{N} m_q F(|\mathbf{r}_p - \mathbf{r}_q|)(\mathbf{r}_p - \mathbf{r}_q) \cdot (\mathbf{v}_p - \mathbf{v}_q), \tag{61}$$

and see Español and Revenga [135] to deduce that

$$(B_i^c)_p = (a\partial_i \nabla \cdot \mathbf{v})_p = a_p \sum_{q=1}^{N} \frac{F(|\mathbf{r}_p - \mathbf{r}_q|)}{\rho_q} \times \left[5\left[\frac{(\mathbf{r}_p - \mathbf{r}_q)^i(\mathbf{r}_p - \mathbf{r}_q) \cdot (\mathbf{v}_p - \mathbf{v}_q)}{(\mathbf{r}_p - \mathbf{r}_q)^2}\right] - (\mathbf{v}_p - \mathbf{v}_q)^i\right], \tag{62}$$

so that

$$B_p^i = (B_i^a)_p \times (B_i^b)_p + (B_i^c)_p. \tag{63}$$

For the pressure gradient term in Eqn. 55, the standard SPH transformation was used [134] to obtain:

$$(\nabla P)_p = -\rho_p \sum_{q=1}^{N} m_q \left( \frac{P_p}{\rho_p^2} + \frac{P_q}{\rho_q^2} \right) F(|\mathbf{r}_p - \mathbf{r}_q|)(\mathbf{r}_p - \mathbf{r}_q) \tag{64}$$

To determine the local value of the fluid viscosity, it will be necessary at first to evaluate the local shear rate tensor. This is discretized as follows:

$$(\dot{\gamma}_{ij})_p = \sum_{q=1}^{N} \frac{F(|\mathbf{r}_p - \mathbf{r}_q|)}{\rho_q / m_q} (\mathbf{r}_p - \mathbf{r}_q)_i (\mathbf{v}_p - \mathbf{v}_q)_j. \tag{65}$$

The local magnitude is then,

$$\dot{\gamma}_p = \sqrt{\frac{\sum_{ij} (\dot{\gamma}_{ij})_p^2}{2}} \tag{66}$$

### 4.3.1 Rigid Body Inclusions

As commonly done in Dissipative Particle Dynamics (DPD)-based simulations (see Hoggerbrugge [130] and Martys [132, 133]), a colloid or nanoparticle embedded within the flow is defined as an assembly of constrained SPH particles so that they form a rigid body. The motion of this rigid body is then determined by summing the interparticle forces due to the neighboring SPH particles (and if required, any other auxiliary forces e.g., lubrication, body forces, colloidal, etc.). Since the volume fraction of the nanoparticle experiments are so low (on the order of .02%), we can safely assume that the nanoparticles, whether they be spherical or sheet like, do not interact with one another. Therefore, the only forces that need

accounting for are the ones provided from the neighboring SPH particles. Once the forces have been summed, the linear and angular acceleration of the inclusion can be determined from its second moment of inertia. The motion of the nanoparticle is then updated along with the motion of the fluid SPH particles at each time step.

### 4.3.2 Stress Tensor

In order to evaluate the rheological properties of the composite fluid (i.e. the fluid matrix with any rigid body inclusions combined), it is necessary to construct a stress tensor similar to that used both in molecular dynamics (MD) and DPD simulations [130, 132, 133, 138]. By first rewriting:

$$m\left(\frac{Dv_i}{Dt}\right)_p = \frac{m}{\rho_p}\left[-\left(\frac{\partial P}{\partial x_i}\right)_p + (A_i)_p + (B_i)_p\right] \tag{67}$$

as

$$m\left(\frac{Dv_i}{Dt}\right)_p = \sum_{q\neq p} f^i_{pq} \tag{68}$$

we can think of each neighboring SPH particle, q, contributing a force $\mathbf{f}_{pq}$ on SPH particle p, as all terms on the right hand side of Eqn. 67 as a result as a summation over all neighboring particles. There are then two contributions to the stress tensor. One stems from the propagation of momentum and inter-particle forces on the SPH particles given by

$$\sigma_{ij} = \frac{1}{V_t}\sum_{q=1}^{N}\frac{1}{m_q}\tilde{p}^q_i\tilde{p}^q_j + \frac{1}{2V_t}\sum_{q=1}^{N} f^i_{pq}(r_p - r_q)_j, \tag{69}$$

where $\tilde{p}^q$ is the momentum of particle $q$ relative to the macroscopic velocity field and $V_t$ is the total volume of the system. The other contribution to the stress tensor arises due to the constraint forces holding the rigid body SPH inclusion together. The constraint forces are determined by accounting for the rigid body motion in the individual particle displacements and the change in velocity at each time step. See Martys and Mountain [139] for a more detailed analysis of the correction for the constraint forces. The viscosity of the entire system is then obtained from evaluation of the total stress tensor as follows

$$\mu = \frac{\sigma_{ij}}{\dot{\gamma}} \tag{70}$$

### 4.3.3 Simulations

The goal of the simulations is to try and determine the total viscosity of the composite fluid matrix with a single inclusion corresponding to the same volume fraction as in experiments. To validate the model, it also is necessary to model the fluid alone and compare the calculated viscosity with experiment. The modeling domain consists of a rectangular shear cell with periodic boundary conditions in all directions except the vertical. The size of the shear cell is $150nm$ in the vertical direction, and $4\mu m$ on either side. To combat any particle deficiency's in the vertical direction (due to the lack of periodic boundary conditions in that direction), a technique used in SPH colloquially called "ghost particles" was utilized. This technique ensures certain field variables, such as particle density, do not artificially approach zero as particles approach the wall. As a particle approached either the top or bottom boundary, a copy (or "ghost") of it is created opposite to the boundary. This mirror copy contains the identical field variables as the initial particle, but opposite velocity.

Furthermore, to apply a constant rate of strain at the boundaries in the vertical direction, Lees-Edwards Allen and Tildesley [138] boundary conditions were utilized (Fig. 28). Once a ghost particle is created, a constant velocity is added to ensure a constant linear rate of strain is applied in the entire domain. Fluid particles within a specified geometry were then

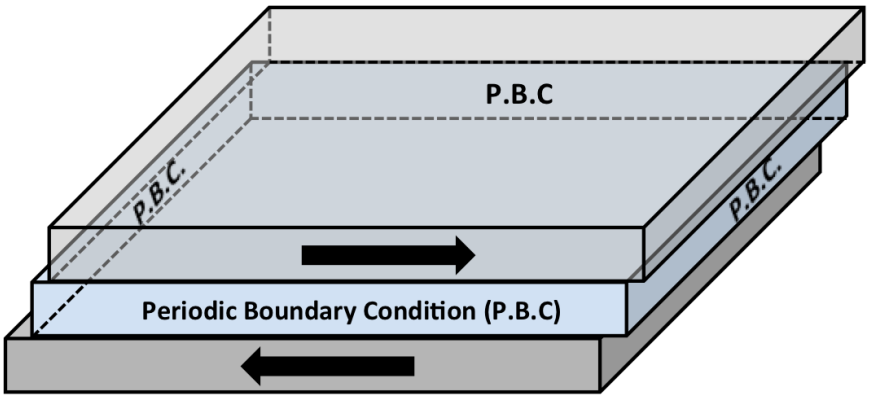


**Figure 28:** Diagram of the computational domain for the rectangular shear cell. The middle box represents the actual domain while the upper and lower boxes represent the Lees-Edwards Allen and Tildesley boundary conditions. Particles within a smoothing distance of either the lower or upper boxes are copied in as a ghost particle but with the added velocity of the box, thus creating a constant shear rate in the middle domain. Regular periodic boundary conditions are used on each of the four surrounding walls.

frozen and allowed to translate and rotate as a rigid body (Fig. 29). A nanosheet geometry was defined as a square with a thickness of $12.5nm$ and a length of $0.3\mu m$. A shear rate of 16000 1/s was applied to the domain and the system was run until steady state conditions were realized.

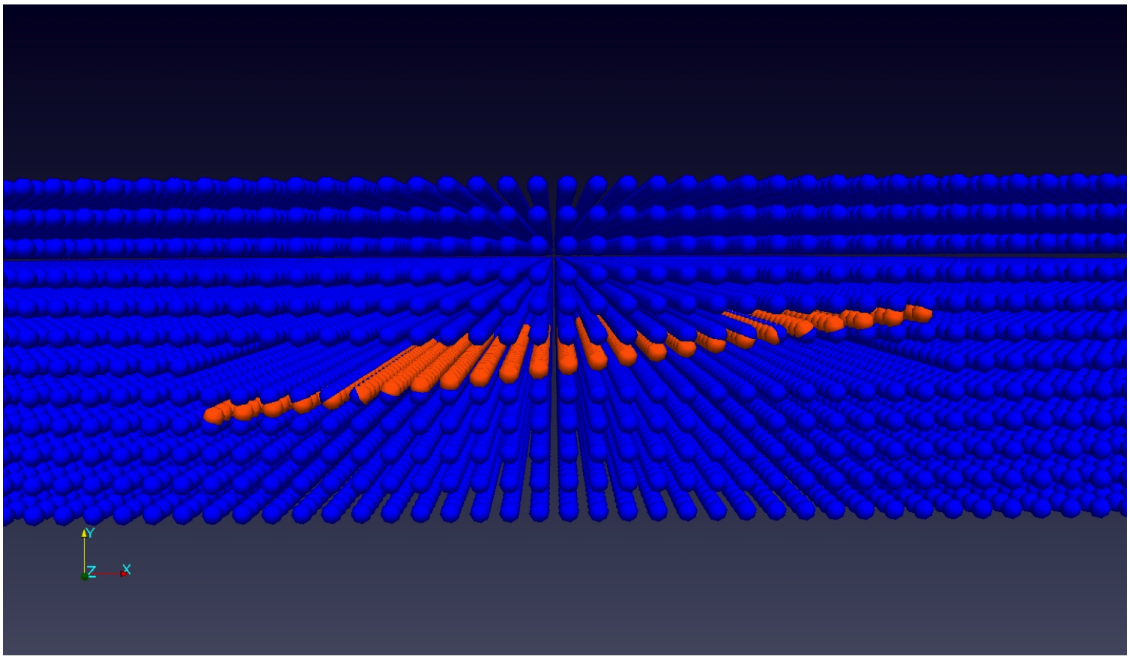

**Figure 29:** Nanosheet Inclusions (in red) submerged in fluid particles (blue)

## 4.4 Results and Discussion

The total viscosity of the fluid composite was calculated and compared with the viscosity of the pure mineral oil (Fig. 30). The viscosity modification was believed to be responsible by the inclination of the $Y_2O_3$ NS in mineral oil under shear. It was noted in the thixotropic study that the viscosity decreased with time in the first 60s [Fig. 27(b)]. The numerical modeling showed an increase. The reason for this discrepancy was that the numerical modeling was on a much shorter time scale ($\sim 0.3ns$) than experimentally achievable ($\sim 60s$) and thus bypassed the thixotropic properties presented in the first $\sim 60s$.

Simulation results (Fig. 31) also showed the local fluid viscosity beneath the NS to be lower than that of the bulk, causing the decrease in the overall viscosity. Since the overall

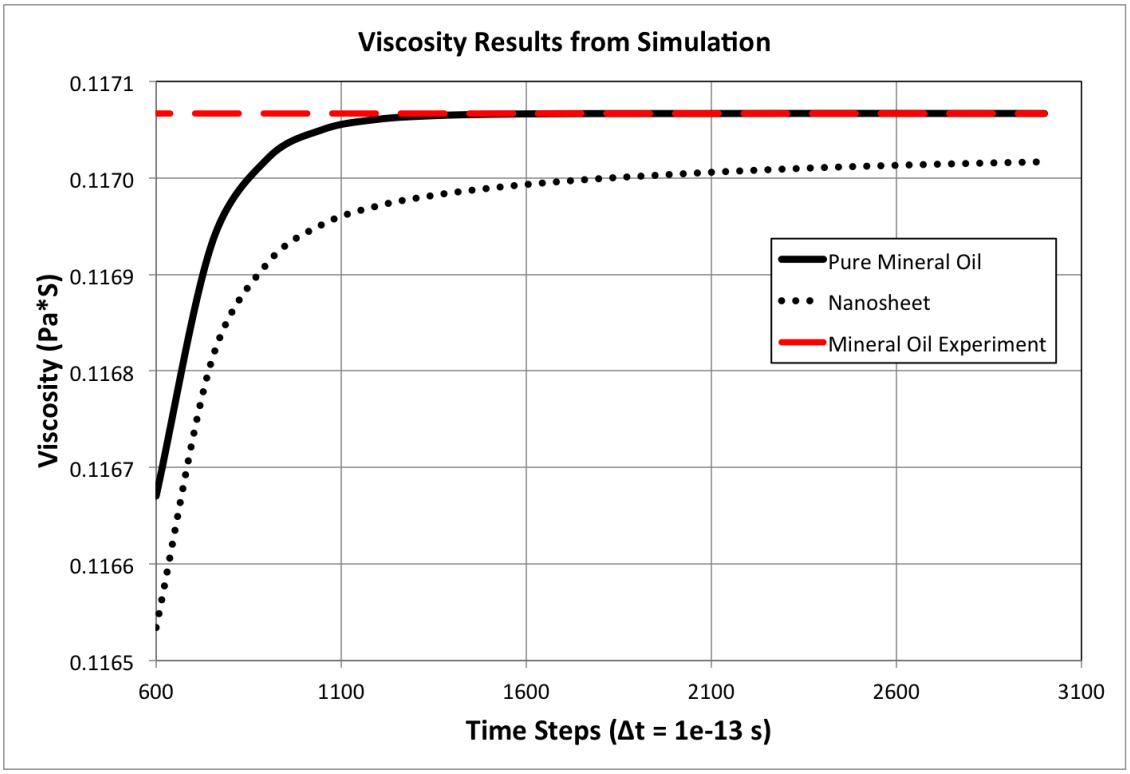


**Figure 30:** Viscosity results from shear cell simulation. Nanosheet shows appreciable viscosity reduction.

viscosity drop stemmed from the decrease in the local fluid viscosity under the NS, the viscosity reduction was originally attributable to the hydrodynamic behavior of the fluid flow surrounding each NS. Another potential reason for the viscosity reduction could include the shear thinning of the suspension fluid or artificial surface tension effects (see Section 7.2). A motivation to include surface tension effects can be found by looking at the dimensionless Eötvös number (or Bond number) [142] defined as follows:

$$Bo = \frac{\rho a L_c^2}{\gamma} \tag{71}$$

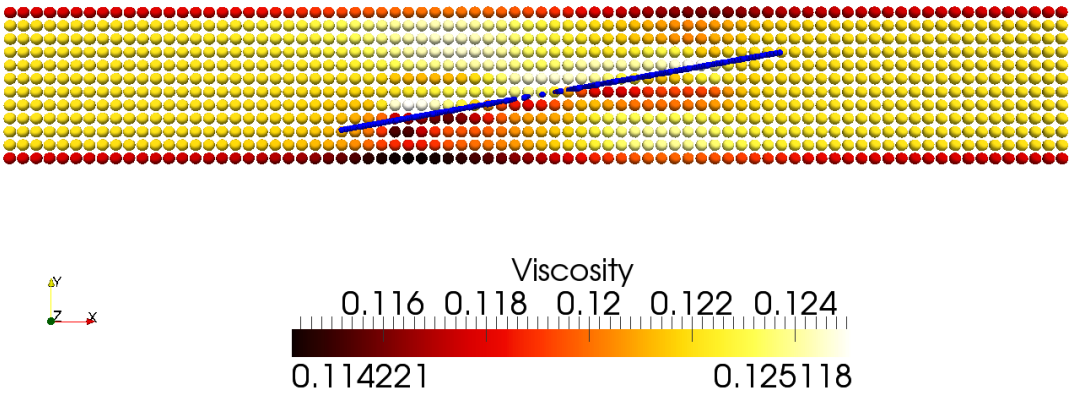


**Figure 31:** Prediction of local viscosity distribution, in $Pa \cdot s$, around the NS from numerical model ($10^o$ inclination) at t = 30 ns.

where $\rho$ is the density of the fluid, $a$ is the acceleration associated with a body force, $L_c$ is the characteristic length scale, and $\gamma$ is the surface tension. By looking at the properties of mineral oil (with $\rho = 5010 kg/m^3$, and $\gamma \approx 30.05 \times 10^{-5} N/cm$[143]) and our characteristic length scale $L_c \approx 100nm$ with an acceleration of gravity, we see we get a Bond value of $\approx 1e-8$, much smaller than one. Therefore, surface tension effects in this tribological interface cannot be ignored.

## 4.5 Conclusion

Sheet-like 2D nanoparticles of $Y_2O_3$ were investigated as an effective lubricant additive. The improvement in lubrication and reduction in viscosity were observed in mineral oil with

the existence of $Y_2O_3$ NS. The 0.1 wt.% of the $Y_2O_3$ NS additive was capable of reducing the viscosity by as much as $\sim 5\%$, respectively. Smoothed particle hydrodynamic based fluid dynamic simulations confirmed the viscosity reduction but with non-physical reasons and without the same magnitude of viscosity reduction as seen with experiment. Specifically, results showed a computational artifact dealing with the density calculation at the interface between the NS and the surrounding mineral oil. It is believed that this contributed to an artificial surface tension at the boundary. While this led to results that compared with experiment, it is relatively obvious that a surface tension model must now be implemented into the simulation to successfully model and explain the non-Einstein like viscosity drop experienced with 2D nanoparticle additives.

# 5 Surface Tension

Surface tension (or surface energy if referring to a solid) is the free energy change $\gamma$ when the surface area of a medium is increased by a unit area. It it closely related to the work of adhesion/cohesion in a vacuum. The work of adhesion, $W_{12}$, is defined as the free energy change, or reversible work done, to separate unit areas of two different media ($1 \neq 2$). While for two identical media ($1 = 2$), it becomes the work of cohesion $W_{11}$. The process of creating a unit area of surface is equivalent to separating two half-unit areas from contact, so that the follow relationship can be made:

$$\gamma_1 = \frac{1}{2} W_{11} \tag{72}$$

For solids, $\gamma_1$ is commonly denoted by $\gamma_s$ and is given in units of energy per unit area ($mJ \cdot m^{-2}$). For liquids, $\gamma_1$ is commonly denoted by $\gamma_L$ and is usually given in units of tension per unit length ($mN \cdot m^{-1}$), which is numerically and dimensionally the same as the surface free energy.

These surface energies are determined from the intermolecular forces between two surfaces, namely van der Waals interactions. These interactions are further composed of three interactions known as the Keesom force (the force between two permanent dipole molecules), the Debye force (between a permanent and induced dipole), and the London dispersion force (two instantaneously induced dipoles). It is this last interaction that makes up perhaps the most important contribution to the total van der Waals force between atoms and molecules because they are always present and play a role in a host of important phenomena such as adhesion; surface tension; physical adsorption; wetting; strengths of solids; flocculation of

particles in liquids; the properties of gases; liquids, and thin films; and the structures of condensed macromolecules such as proteins and polymers [144].

Dispersion forces are quantum mechanical in origin and can take a host of theoretical treatments of varying complexity, the most rigorous of which would have to rely with quantum electrodynamics. While these forces are quantum mechanical (in determining the instantaneous, but fluctuating, dipole moments of neutral atoms), the resulting interaction is essentially still electrostatic, and can therefore be modeled as any other classical force.

At very small interatomic distances, the electron clouds of atoms overlap, and there arises a strong repulsive force that determines how close two atoms or molecules can ultimately approach each other. The total intermolecular pair potential is obtained by summing the attractive and repulsive potentials. The best known of these is the Lennard-Jones, L-J, or "6-12" potential, similar to Eq. 22 which is widely used because of its simplicity and inverse sixth-power attractive van der Waals term.

## 5.1 SPH: Modeling Surface Tension

The SPH discretization scheme reduces the Navier-Stokes equation to a system of ordinary differential equations that have the form of Newton's Second Law of motion for each particle (see Section 1). This simplicity allows a variety of physical and chemical effects to be incorporated into SPH models with relatively little code-development effort through pair-wise molecular type interactions. Morris [145] modeled surface tension based on its macroscopic description with surface tension forces that were proportional to the fluid-fluid interface curvature. This approach gives an accurate estimation of the effects of surface

tension but involves rather complex calculations of front curvatures that, in some cases, may lead to significant errors. Nugent and Posch [146] used attractive forces, corresponding to the cohesive pressure in the van der Waals equation of state, to simulate surface tension in two-dimensional SPH simulations. While this technique works relatively well, it comes at the computational cost of increasing the range of the attractive force to at least twice the range of the SPH weighting function for stability. This results in a significant increase in the number of particle interactions and thus in computer time to perform a simulation.

Tartakovsky and Meakin [147, 148] utilized a combination of short-range repulsive and (relatively) long-range attractive particle-particle interactions (with the range of the attractive interactions equal to the range of SPH weighting function) with standard SPH equations. The use of a combination of short-range repulsive and long-range attractive interactions was motivated by the molecular origins of surface tension as briefly summarized in the previous subsection. Incorporating particle-particle interaction forces into the SPH model allows one to simulate not only surface tension, but also fluid-solid interactions, resulting in well-defined fluid-solid contact angles under both static and dynamic conditions. This is crucial since there is no analytical relationship between these interparticle forces and emergent properties such as surface tension; therefore numerical experiments must be performed in which well-defined fluid-solid contact angles are required to calculate physical properties of the system. The consistency of this model with modeling surface tension (Tartakovsky and Meakin performed four different numerical experiments resulting with the same value of surface tension), coupled with its relative simplicity, is why it was chosen to augment the nanosheet simulations presented in the previous section.

As mentioned in Section 4.4, Hoover [140] and Colagrossi and Landrini [141] used SPH

to model the immiscible flow of two fluids and found that the standard SPH formulation of Gingold and Monaghan [33] creates artificial surface tension on the boundary between two fluids or a fluid/solid interface. Tartakovsky and Meakin [148] showed that this artificial surface tension can be eliminated by using SPH equations based on the particle number density instead of the particle fluid density used in standard SPH formulations (see Section 1). Specifically, in Eqn. 15, $\rho_i/m_i$ can be replaced by the particle number density $n_i$,

$$n_i = \frac{\rho_i}{m_i} \tag{73}$$

and Eqn. 15 can be replaced by

$$f(x_i) \approx \sum_{j=1}^{N} \frac{f(x_j)}{n_j} \cdot W(x_i - x_j, \lambda) \tag{74}$$

where $n_i$ has units of $(L^{-3})$ and satisfies

$$n_i \Delta V_i \equiv 1. \tag{75}$$

The particle number density calculated from (74) is

$$n_i = \sum_{j=1}^{N} W(x_i - x_j, \lambda), \tag{76}$$

and gradients can be calculated from

$$\nabla f(x_i) \approx \sum_{j=1}^{N} \frac{f(x_j)}{n_j} \cdot \nabla_i W(x_i - x_j, \lambda). \tag{77}$$

The motion of each particle is governed by momentum and mass (or particle number) conservation principles described by the Navier-Stokes equations. Expressions 74 and 77 are used to approximate the fields and spatial derivatives that enter into the Navier-Stokes equations. For simple single-phase flows, for which all the particles are assumed to have the same time invariant mass, the same results are obtained using either $n_i$ or $\rho$ in the SPH equations. For multiphase flows, it is important to distinguish between the particles masses and particle densities. The modified SPH equations, based on the particle number density, do not generate artificial surface tension. We can then proceed to add a model for surface tension which will allow better control over its magnitude rather than depend on the density gradient across a interface.

## 5.2 Fluid-Solid Interactions

Particle-particle interactions are used to prevent the different fluids from mixing, and this generates a surface tension. Immiscible fluids usually have different wetting properties, so that one of the fluids will preferentially wet solid boundaries in the presence of another fluid. The advantage of the discretized SPH momentum conservation equation is its simple physical interpretation. It has exactly the same form as Newton's Second Law of motion, and the total force acting on SPH particle $i$ is the sum of the forces acting between particle $i$ and the surrounding particles.

Fluid-solid and fluid-fluid interactions (that are a physical consequence of molecule interactions) can be accounted for by adding a molecule-like pair-wise particle-paricle interaction

into Eq.68 leading to the particle equation of motion

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i + \mathbf{F}_i^{interaction}, \tag{78}$$

where $\mathbf{F}_i^{interaction}$ is the force acting on particle $i$ due to the particle-particle interactions. The interaction force between pair of particles $i$ and $j$ is given by

$$\mathbf{F}_{ij} = \begin{cases} s_{ij} \cos\left(\frac{1.5\pi}{3\lambda}|\mathbf{r}_j - \mathbf{r}_i|\right) \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|} & |\mathbf{r}_j - \mathbf{r}_i| \leq 3\lambda, \\ 0 & |\mathbf{r}_j - \mathbf{r}_i| > 3\lambda, \end{cases} \tag{79}$$

where $s_{ij}$ is the strength of the force. The total force to interparticle interactions acting on any particle $i$ is given by

$$\mathbf{F}_i^{interaction} = \sum_{j=1}^{N} \mathbf{F}_{ij}. \tag{80}$$

It should also be mentioned that since $\mathbf{F}_{ij} = -\mathbf{F}_{ji}$, the particle-particle interactions conserve momentum exactly as should be expected. As will be mentioned later, the exact form of the particle-particle interactions is not critical to the success of the simulations, so long as the interactions are repulsive at short distance and attractive at large distances. Numerical experiments can then be performed to correlate the particle-particle forces (namely their magnitudes) to experimental values for surface tension. In addition, computational efficiency requires a long distance interaction force cutoff (at $\kappa\lambda$ in this case) to reduce the number of particle-particle interactions that must be calculated.

For a given fluid, the magnitude of this force depends only on the relative distance

between particles. The force is repulsive for distances less than $\lambda/3$, attractive for distances between $\lambda/3$ and $\lambda$, and zero for distances larger than $\lambda$. Apart from the effects of small density and configuration fluctuations in the interior of the fluids, the total particle-particle interaction force acting on the fluid particles is non-zero only near fluid-fluid interfaces and fluid-solid interfaces. By varying the the strength of the force between like particles and particles of different species, one can tune surface tension effects between two materials to ones liking.

At each time step in a simulation, the particle number densities, $n_i$, at each of the particles are calculated using Eqn. 76, and the pressure at each particle is obtained using the equation of state [40]:

$$P_i = P_{eq} n_i / n_{eq}, \tag{81}$$

where $n_{eq}$ and $P_{eq}$ are the equilibrium density and pressure.

## 5.3 Surface Tension Numerical Experiment

For many types of mesh free methods, there exists no analytical relationship between interparticle interactions and surface tension calculations (the exception is the Lattice Boltzmann method if the model is based on a Landau-Ginzburg and Cahn-Hilliard type of free energy functional) [149]. However, if the more popular particle-particle interaction approach is used [150], the surface tension of the lattice Boltzmann fluid must be measured using approaches similar to those applied to the SPH model described below.

In the modified SPH model, the surface tension is not prescribed explicitly. Instead, it is

determined by the interaction forces and the equation of state described in Section 5.1. By varying the magnitude of the interaction force in Eq. 79 between particles of the same and different species, one can define the surface tension between an interface. One can then run parametric numerical experiments until the calculated emergent surface tension properties are similar to experiment.

Figure 32 depicts the contact angle, $\theta_c$, that the drop of mineral oil makes with a substrate coated with $Y_2O_3$. Based on the experimental values for the surface energy of $Y_2O_3$ ($\gamma_S = 2.161 J/m^2$) [151], the surface tension of mineral oil ($\gamma_L = 0.0303 N/m$) [143], and the value of the contact angle ($\theta_c = 50^o$), we can use the Young equation:

$$\gamma_{SL} = \gamma_S - \gamma_L \cos(\theta_c) \tag{82}$$

to calculate the interfacial energy between the mineral oil and the $Y_2O_3$ ($\gamma_{SL} = 2.142 J/m^2$).

The numerical surface tension can be determined from the equilibrium radius of a liquid drop and the difference between the pressure in its interior, $P_0$, and the pressure in the surrounding fluid $P_l$, using the Young-Laplace equation:

$$P_0 - P_l = \sigma / R. \tag{83}$$

To calculate the interfacial energy between the mineral oil and $Y_2O_3$ surface, a simulation was run in which a three-dimensional computational domain ($25 \times 25 \times 25$ in units of $\lambda$) was set up with fluid particles with the same mass, density, and viscosity as mineral oil. Then,

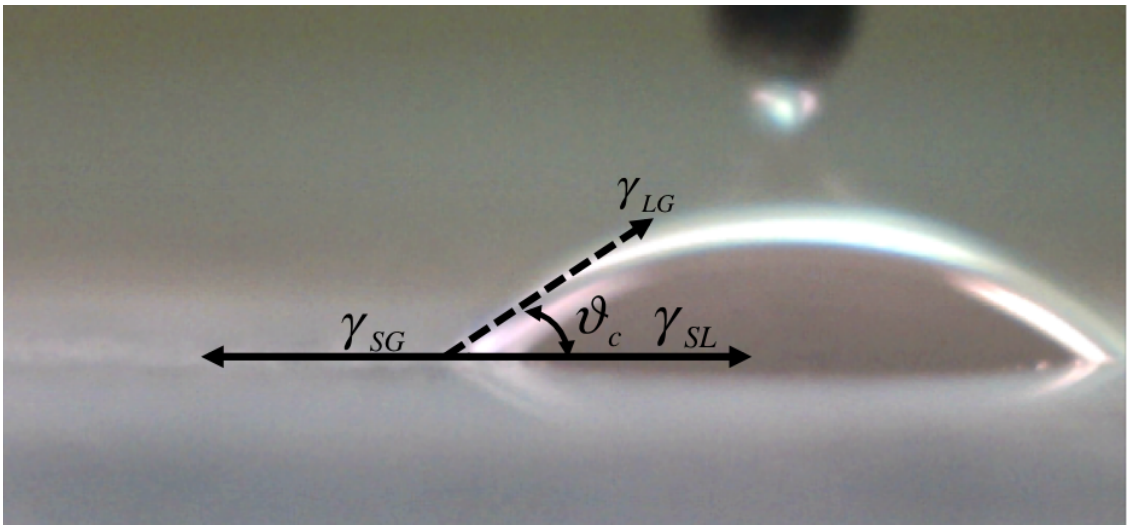


**Figure 32:** Experimental contact angle between a drop of mineral oil and substrate coated with $Y_2O_3$. $\gamma_{SL}$ is the surface tension between the solid and the liquid, $\gamma_{LG}$ between the liquid and vapor, and $\gamma_{SG}$ between the solid and vapor.

the particles with coordinates

$$(x_i - 12.5)^2 + (y_i - 12.5)^2 + (z_i - 12.5)^2 > R_0^2 \tag{84}$$

were chosen to represent a liquid with the mass and density of $Y_2O_3$ (as shown in Fig. 33). Then, Eq. 78 was applied to the particle system with the amplitudes of the forces acting between particles of the same fluids set to $s_{11} = s_{22} = 1e - 10$ and the amplitude of the forces acting between particles representing different fluids set to $s_{12} = 72e - 10$.

Because of the particle-particle interactions $\mathbf{F}_{ij}$, the pressure inside and outside the droplet cannot be obtained directly from the equation of state. However, Hoover [140] has shown that SPH is isomorphic with molecular dynamics with many-body particle-particle interactions. This allows the SPH equations and the particle-particle interactions to be treated

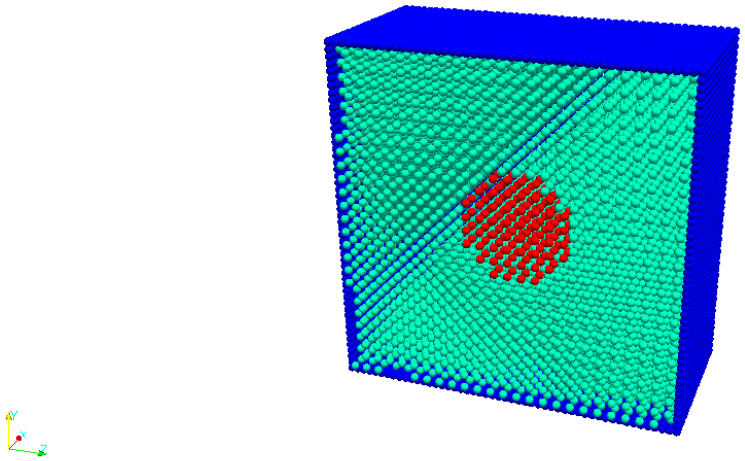

**Figure 33:** A cutaway view of the SPH model of liquid drop numerical experiment. The blue particles represent repulsive wall particles, the green particles are the $Y_2O_3$ particles, and the red particles represent the mineral oil.

in a consistent manner so that the pressure can be calculated from the *total* particle-particle interaction forces [138] using the virial theorem relationship

$$P_f = \frac{1}{4V} \sum_i \sum_j \mathbf{r}_{ij} \cdot \mathbf{f}_{ij}, \tag{85}$$

where

$$\mathbf{f}_{ij} = -\left( \frac{P_j}{n_j^2} + \frac{P_i}{n_i^2} \right) \nabla_i W(\mathbf{r}_i - \mathbf{r}_j, \lambda) + \mathbf{F}_{ij}. \tag{86}$$

The summation in Eq. 85 is over all particles in a computational domain occupying a volume $V$ and self-interactions ($i = j$) are excluded. Figure 34 shows the pressure difference $P_0 - P_l$ versus $1/R$, and a interfacial surface tension of approximately $2.156 J/m^2$ was found from this data using Eq. 83 which is similar to the experimental value determined from Eq. 82. These

parameters can then be implemented into the nanosheet simulations to properly incorporate surface tension effects.

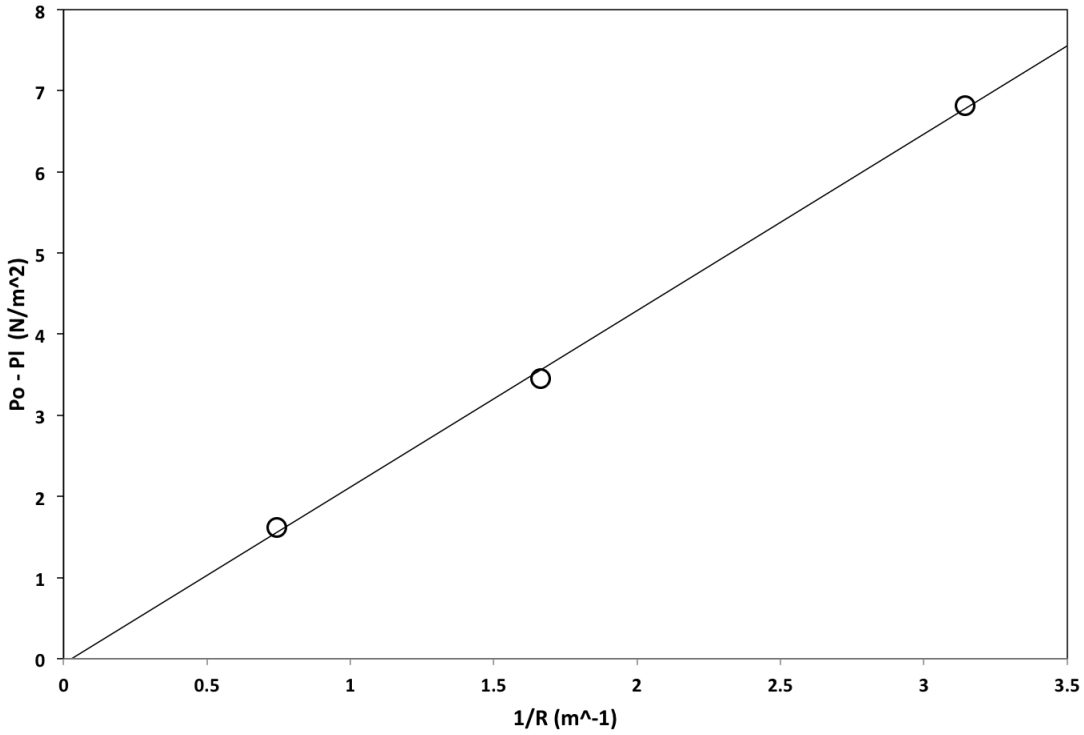


**Figure 34:** Difference in pressure inside and outside of liquid bubble $P_o - P_l$ versus $1/R$. The slope of the line represents a best fit value for the surface tension over three numerical experiments.

## 5.4 Results and Discussion

With the surface tension physics now implemented into the SPH model, the nanosheet simulations as discussed in Section 4 can now be run and compared with experiment. Figure 35 depicts the total viscosity of the fluid composite and compares it with the viscosity of the pure mineral oil. It is thus shown that a sufficient physical explanation for the viscosity

drop seen earlier in the numerical simulations, as well as with experiment, can be attributed to surface interfacial energy between the nanosheet and the fluid matrix. This effect is competing with the routine hydrodynamic Einstein-like effect of increasing the viscosity, however with 2D nanomaterials such as the $Y_2O_3$ nanosheet, the dominant effect seems to be the surface tension. This can be readily explained for the relatively high surface area of the nanosheet when compared to its overall volume. Once nanoparticles become large enough to where their surface area to volume ratio reduces to more macroscopic values, Einstein's relation dominants and a viscosity increase is still to be expected. Figure 36 depicts the local viscosity distribution surrounding the nanosheet with surface tension effects included. One can see that that local particle viscosity decreases above and below the nanosheet due to surface tension effects.

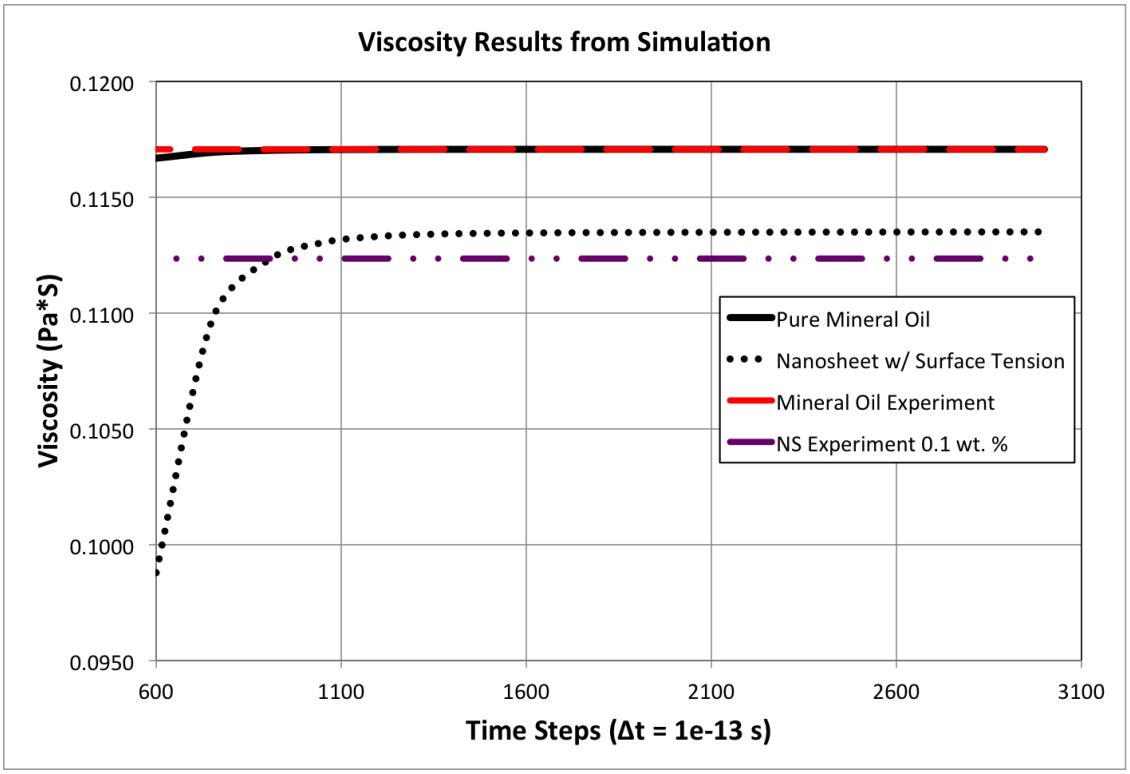


**Figure 35:** Viscosity results from shear cell simulations after surface tension effects were added. The nanosheets shows a more appreciable reduction in viscosity than before and is more comparable with experiment.

# 6 Conclusion

Smoothed Particle Hydrodynamics (SPH) is a mesh free, Langrangian, particle-based method initially created to model astrophysical phenomena. Since its introduction, it has been extended to applications in continuum solid and fluid mechanics. As opposed to meshed techniques, SPH discretizes the continuum into a set of discrete particles. These particles have no connectivity between them, except for interparticle interactions dictated by continuum, constitutive relations. SPH has been shown to be suitable for modeling moving or

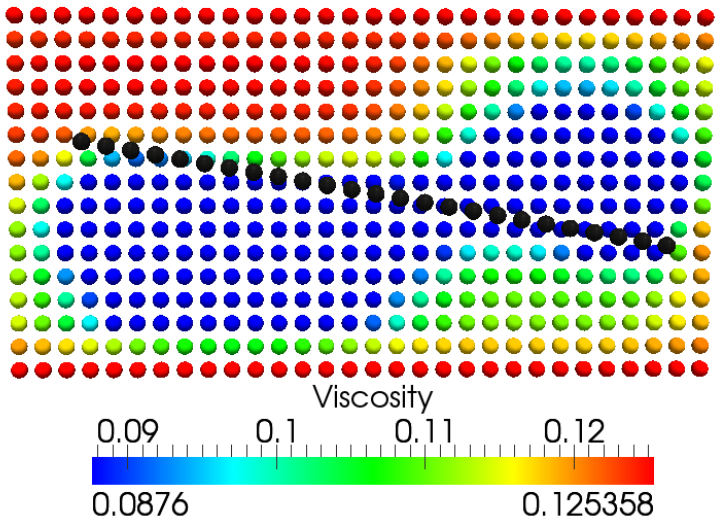


**Figure 36:** Prediction of local viscosity distribution, in $Pa \cdot s$, around the NS from the numerical model with surface tension effects included. The NS is represented with the black particles while the other particles represent the fluid.

deforming boundaries, multiphase fluids, and free surfaces.

Section 2 showed how an in-house solver was created in order to simulate hydrodynamic lubrication utilizing the SPH method. Transient hydrodynamic lubrication in a pad bearing geometry was modeled, and the results were validated by comparison to computational fluid dynamics (CFD) and an analytical solution provided by lubrication theory. Results for the pressure distribution between SPH and CFD were agreeable.

Section 3 showed how SPH can handle more complicated and dynamic fluid-solid interactions. A study was conducted in modeling fluid flow between meshing involute gear teeth. For further applicability to industry, particulate motion within the flowfield was also studied. The fluid phase flowfield was obtained by solving the Navier-Stokes equations while particle trajectories were calculated by integrating a drag force equation of motion. It was demon-

strated how the particle size is the dominant factor in determining the particle entrainment during a gear meshing cycle.

Section 4 introduced the unusual, non-Einstein-like rheological properties of nanoparticle additives. Experimental evidence was shown on how 2D yttrium oxide ($Y_2O_3$) nanosheet (NS) additives can improve the mechanical and transport phenomena of mineral oil. A SPH model was introduced that could model both the non-Newtonian, shear thinning characteristics of the fluid matrix while simultaneously tracking the fluid-solid interface between the nanosheet and the fluid. Initial results showed a viscosity reduction with the addition of the NS, however it was later discovered that this effect was due to a computational artifact with the SPH density summation method. This artifact produced an artificial surface tension that was not intentionally modeled and led to the decrease in viscosity. This thus prompted a surface tension model be added to the SPH method to properly account for these effects.

Section 5 talked about the effects of surface tension and energy between fluids and solids. An SPH model was introduced that could incorporate surface tension effects and fluid-solid interactions. Numerical experiments were performed to validate the model and compare it with experimental evidence of the surface tension between the mineral oil and $Y_2O_3$ nanosheet. Results showed that when the surface tension effects were successfully implemented, a reduction in viscosity more comparable to experimental evidence was noted.

This work has introduced the SPH method to the field of tribology and exhibits the flexibility and versatility with this computational method. SPH is advantageous in modeling complex geometries, interfaces, liquids, and also in incorporating different physics when needed into a model. This research has also shown how computational simulations can be utilized to verify experimental rheological results with nanoadditive particles. With these

results, next generation lubricants can be simulated with various additive materials and geometries for specific applications and conditions.

# References


[1] Albert Einstein. On the theory of brownian movement. *Ann. Phys. (Leipz.)*, 19: 371–381, 1906.

[2] J. C. VanDerWerff and C.G. de Kruif. Hard-sphere colloidal dispersions: the scaling of rheological properties with particle size, volume fraction, and shear rate. *J. Rheol.*, 33:421–454, 1989.

[3] W.B. Russel, D.A. Saville, and W.R. Schowalter. *Colloidal Dispersions*. Cambridge Univ. Press, 1989.

[4] A.B. Metzner. Rheology of suspensions in polymeric liquids. *J. Rheol.*, 29:739–775, 1985.

[5] Ir.R. Rutgers. Relative viscosity of suspensions of rigid spheres in newtonian liquids. *Rheol Acta*, 2(3):202–210, 1962.

[6] V.V. Jinescu. The rheology of suspensions. *International Chemical Engineering*, 14 (397-420), 1974.

[7] M.D. Croucher and T.H. Milkie. *The Effect of Polymers on Dispersion Properties*. Academic Press, 1982.

[8] M.R. Kamal and A. Mutel. Rheological properties of suspensions in newtonian and non-newtonian fluids. *J. Polym. Eng.*, 5(4):293–382, 1985.

[9] G.K. Batchelor. The effect of brownian motion on the bulk stress in a suspension of spherical particles. *Journal of Fluid Mechanics*, 83:97–117, 1977.

[10] P. Mazur and W. Van Saarloos. Many-sphere hydrodynamic interactions and mobilities in a suspension. *Physica A*, 115A(1-2):21–57, 1982.

[11] C.W.J. Beenakker. The effective viscosity of a concentrated suspension of spheres (and its relation to diffusion). *Physica A*, 128:48–81, 1984.

[12] C.W.J. Beenakker and P. Mazur. Diffusion of spheres in a concentrated suspension. ii.

*Physica A*, 126:349–70, 1984.

[13] B. Cichocki and B.U. Felderhof. Short-time diffusion coefficients and high frequency viscosity of dilute suspensions of spherical brownian particles. *Journal of Chemical Physics*, 89(2):1049–1055, 1988.

[14] H.J.H. Clercx and P.P.J.M. Schram. High-frequency effective viscosity of hard-sphere suspensions. *Physical Review A*, 45(2):860–872, 1992.

[15] G.K. Batchelor. The stress system in a suspension of force-free particles. *Journal of Fluid Mechanics*, 41(3):545–570, 1970.

[16] G.K. Batchelor and J.T. Green. Determination of the bulk stress in a suspension of spherical particles to order $c^2$. *Journal of Fluid Mechanics*, 56(3):401–427, 1972.

[17] G.K. Batchelor and J.T. Green. Hydrodynamic interaction of two small freely-moving spheres in a linear flow field. *Journal of Fluid Mechanics*, 56(2):375–400, 1972.

[18] W.B. Russel and A.P. Gast. Nonequilibrium statistical mechanics of concentrated colloidal dispersions: hard spheres in weak flows. *Journal of Chemical Physics*, 84(3): 1815–1826, 1986.

[19] D. Bedeaux, R. Kapral, and P. Mazur. The effective shear viscosity of a uniform suspension of spheres. *Physica A*, 88A(1):88–121, 1977.

[20] I.M. Krieger and T.J. Dougherty. A mechanism for non-newtonian flow in suspensions of rigid spheres. *Transactions of the Society of Rheology*, 3:137–152, 1959.

[21] D. Quemada. *Lecture Notes in Physics: Stability of Thermodynamic Systems*. Springer, 1984.

[22] A.J.C. Ladd. Hydrodynamic transport coefficients of random dispersions of hard spheres. *Journal of Chemical Physics*, 93(5):3484–3494, 1990.

[23] G. Bossis and J.F. Brady. The rheology of brownian suspensions. *Journal of Chemical Physics*, 91(3):1866–1874, 1989.

[24] H.A. Barnes, M.F. Edwards, and L.V. Woodcock. Applications of computer simultations to desnse suspension rheology. *Chemical Engineering Science*, 42(4):591–608, 1987.

[25] G. Bossis and J.F. Brady. Dynamic simulation of sheared suspensions. i. general method. *Journal of Chemical Physics*, 80(10):5141–5154, 1984.

[26] L.V. Woodcock. Origins of thixotropy. *Physical Review Letters*, 54(14):1513–1516, 1985.

[27] D.J. Evans. Rheology and thermodynamics from nonequilibrium molecular dynamics. *International Journal of Thermophysics*, 7(3):573–584, 1986.

[28] S. Hess. Shear-induced melting and reentrant positional ordering in a system of spherical particles. *International Journal of Thermophysics*, 6(6):657–671, 1985.

[29] S. Jain, J.G.P. Goossens, G.W.M. Peters, M.V. Duin, and P.J. Lemstra. Strong decrease in viscosity of nanoparticle-filled polymer melts through selective adsorption. *Soft matter*, 4(9):1848, 2008.

[30] M.E. Mackay, T.T. Dao, A. Tuteja, D.L. Ho, B.V. Horn, H.C. Kim, and C.J. Hawker. Nanoscale effects leading to non-einstein-like decrease in viscosity. *Nature Materials*, 2:762–766, 2003.

[31] Y. Tan, Y. Song, and Q. Zheng. Hydrogen bonding-driven rheological modulation of chemically reduced graphene oxide/poly(vinyl alcohol) suspensions and its application in electrospinning. *Nanoscale*, 4(22):6997, 2012.

[32] S.P. Thomas, S.A. Girei, M.A. Atieh, S.K. De, and A. Al-Juhani. Rheological behavior of polypropylene nanocomposites at low concentration of surface modified carbon nanotubes. *Polymer Engineering and Science*, 52(9):1868–1873, 2012.

[33] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, (181):375–389, 1977.

[34] L.B. Lucy. Numerical approach to testing the fission hypothesis. *Astronomical Journal*,

(82):1013–1024, 1977.

[35] Carla Antoci, Mario Gallati, and Stefano Sibilla. Numerical simulation of fluid-structure interaction by sph. *Computers and Structures*, (85):879–890, 2007.

[36] W. Benz and E. Asphaug. Simulations of brittle solids using smooth particle hydrodynamics. *Computer Physics Communications*, (87):253–265, 1995.

[37] J.P. Gray, J.J. Monaghan, and R.P. Swift. Sph elastic dynamics. *Computer Methods in Applied Mechanics and Engineering*, (190):6641–6662, 2001.

[38] L.D. Libersky, A.G. Petschek, T.C. Carney, J.R. Hipp, and F.A. Allahdadi. High strain lagrangian hydrodynamics. *Journal of Computational Physics*, (109):67–75, 1993.

[39] J.J. Monaghan. Simulating free surface flows with sph. *Journal of Computational Physics*, 110:399–406, 1994.

[40] J.P. Morris, P.J. Fox, and Y. Zhu. Modeling low reynolds number incompressible flows using sph. *Journal of Computational Physics*, (136):214–226, 1997.

[41] H. Takeda, S.M. Miyama, and M. Sekiya. Numerical simulation of viscous flow by smoothed particle hydrodynamics. *Progress of Theoretical Physics*, (92):939–960, 1994.

[42] M.S. Shadloo, A. Zainali, H. Sadek, and M. Yildiz. Improved incompressible smoothed particle hydrodynamics method for simulating flow around bluff bodies. *Computer Methods in Applied Mechanics and Engineering*, 200:1008–1020, 2011.

[43] G.R. Liu and M.B. Liu. *Smoothed Particle Hydrodynamics: a meshfree particle method*. World Scientific Publishing Co. Pte. Ltd., 2003.

[44] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100(32), 1928.

[45] D.A. Fulk. A numerical analysis of smoothed particle hydrodynamics. *Ph.D. Thesis, Air Force Institute of Technology*, 1994.

[46] L. Hernquist and N. Katz. Treesph-a unification of sph with the hierarchical tree

method. *The Astrophysical Journal Supplement Series*, 70:419–446, 1989.

[47] J.J. Monaghan. Why particle methods work (hydrodynamics). *SIAM Journal on Scientific and Statistical Computing*, 3:422–433, 1982.

[48] J.J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(543), 1992.

[49] J.J. Monaghan and J.C. Lattanzio. A refined particle method for astrophysical problems. *Astronomy Astrophysics*, 149(135), 1985.

[50] I.J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quarterly of Applied Mathematics*, 4(45), 1946.

[51] P.W. Randles and L.D. Libersky. Smoothed particle hydrodynamics some recent improvements and applications. *Computer Methods in Applied Mechanics and Engineering*, 138:375–408, 1996.

[52] J.J. Monaghan and R.A. Gingold. Shock simulation by the particle method of sph. *Journal of Computational Physics*, 52:374–381, 1983.

[53] J.J. Monaghan. Particle methods for hydrodynamics. *Computer Physics Report*, 3: 71–124, 1985.

[54] R.W. Hockney and J.W. Eastwood. *Computer simulations using particles*. Adamhilger, 1988.

[55] C.E. Rhoades. A fast algorithm for calculating particle interactions in smooth particle hydrodynamic simulations. *Computer Physics Communications*, 70:478–482, 1992.

[56] J.C. Simpson. Numerical techniques for three-dimensional smoothed particle hydrodynamics simulations: applications to accretion disks. *The Astrophysical Journal*, 448: 822–831, 1995.

[57] F. Spinato, P.J. Tavner, G.J.W. van Bussel, and E. Koutoulakos. Reliability of wind turbine subassemblies. *IET Renewable Power Generation*, 3(4):387–401, 2009.

[58] R.H. Buckholz. Effect of lubricant inertia near the leading edge of a plane slider bearing. *Journal of Tribology*, 109(1):60–65, 1986.

[59] Jiaxin Zhao, Farshid Sadeghi, and Michael H Hoeprich. Analysis of ehl circular contact start up: Part i-mixed contact model with pressure and film thickness results. *Journal of Tribology*, 123(1):67–74, 2001.

[60] RP Glovnea and HA Spikes. Elastohydrodynamic film formation at the start-up of the motion. *Journal of Engineering Tribology*, 215(2):125–138, 2001.

[61] RP Glovnea and HA Spikes. The influence of lubricant upon ehd film behavior during sudden halting of motion. *Tribology Transactions*, 43(4):731–739, 2000.

[62] N Ren, D Zhu, and SZ Wen. Experimental method for quantitative analysis of transient ehl. *Tribology International*, 24(4):225–230, 1991.

[63] M. J. A. Holmes, H. P. Evans, and R. W. Snidle. Comparison of transient ehl calculations with start-up experiments. In D. Dowson, M. Priest, G. Dalmaz, and A. A. Lubrecht, editors, *Tribology Series*, volume 41, pages 79–89. Elsevier, 2003.

[64] K. Herrebrugh. Elastohydrodynamic squeeze films between two cylinders in normal approach. *Journal of Lubrication Technology*, 92:292 – 302, 1970.

[65] H. Christensen. Elastohydrodynamic theory of spherical bodies in normal approach. *Journal of Lubrication Technology*, 92(145-154), 1970.

[66] Jiaxin Zhao, Farshid Sadeghi, and Michael H. Hoeprich. Analysis of ehl circular contact start up: Part i—mixed contact model with pressure and film thickness results. *Journal of Tribology*, 123(1):67, 2001.

[67] Jiaxin Zhao and Farshid Sadeghi. The effects of a stationary surface pocket on ehl line contact start-up. *Journal of Tribology*, 126(4):672–680, 2004.

[68] G Popovici, CH Venner, and PM Lugt. Effects of load system dynamics on the film thickness in ehl contacts during start up. *Journal of Tribology*, 126(2):258–266, 2004.

[69] C. Fred Higgs, Sum Huan Ng, Len Borucki, Inho Yoon, and Steven Danyluk. A

mixed-lubrication approach to predicting cmp fluid pressure modeling and experiments. *Journal of the Electrochemical Society*, 152(3):193–198, 2005.

[70] Xiaoqing Jin, Leon M. Keer, and Qian Wang. A 3d ehl simulation of cmp: Theoretical framework of modeling. *Journal of the Electrochemical Society*, 152(1):7–15, 2005.

[71] S. Li and A. Kahraman. Prediction of spur gear mechanical power losses using a transient elastohydrodynamic lubrication model. *Tribology Transactions*, 53:554 – 563, 2010.

[72] E. J. Terrell and C. F. Higgs. A modeling approach for predicting the abrasive particle motion during chemical mechanical polishing. *Journal of Tribology*, 129(4):933–41, 2007.

[73] E. J. Terrell and C. F. Higgs III. Hydrodynamics of slurry flow in chemical mechanical polishing. *Journal of the Electrochemical Society*, 153(6):15–22, 2006.

[74] Z. Luan and M. M. Khonsari. A note on the lubricating film in hydrostatic mechanical face seals. *Proceedings of the Institution of Mechanical Engineers, Part J (Journal of Engineering Tribology)*, 222(J4):559–67, 2008.

[75] Sum Huan Ng, Len Borucki, C. Fred Higgs III, Inho Yoon, Andres Osorno, and Steven Danyluk. Tilt and interfacial fluid pressure measurements of a disk sliding on a polymeric pad. *Journal of Tribology*, 127(1):198–205, 2005.

[76] Nadir Patir and H. S. Cheng. Application of average flow model to lubrication between rough sliding surfaces. *Journal of Lubrication Technology*, 101(2):220–230, 1979.

[77] B. Nisson and P. Hansbo. Weak coupling of a reynolds model and a stokes model for hydrodynamic lubrication. *International Journal for Numerical Methods in Fluids*, 66: 730–741, 2010.

[78] L.G. Leal. *Advanced Transport Phenomena: Fluid Mechanics and Convective Transport Processes*. Cambridge University Press, 2007.

[79] Scott Bair and Farrukh Qureshi. Accurate measurements of pressure-viscosity behavior in lubricants. *Tribology Transactions*, 45(3):390–390, 2002.

[80] O. Pinkus and B. Sternlicht. *Theory of Hydrodynamic Lubrication*. McGraw-Hill, New York, 1961.

[81] A.Z. Szeri, A.A. Romandi, and A. Giron-Duarte. Linear force coefficients for squeeze film dampers. *Journal of Lubrication Technology*, 105(F):326–334, 1983.

[82] L. San Andrés and J.M. Vance. Effects of fluid inertia and turbulence on the force coefficients for squeeze film dampers. *Journal Engineering Gas Turbines and Power*, 108:332–339, 1986.

[83] H.G. Elrod, I. Anwar, and R. Colsher. Transient lubricating films with inertia. *Journal of Lubrication Technology*, 105(3):369–374, 1983.

[84] H. Hashimoto, S. Wada, and M. Sumitomo. The effects of fluid inertia forces on the dynamic behavior of short journal bearings in superlaminar flow regime. *Journal of Tribology*, 110(3):539–547, 1988.

[85] J. Tichy and B. Bou-Saïd. Hydrodynamic lubrication and bearing behavior with impulsive loads. *Tribology Transactions*, 34(4):505–512, 1991.

[86] E.R. Booser and Donald F. Wilcock. Hydrodynamic lubrication. *Lubrication Engineering*, 47(8):645–647, 1991.

[87] G.W. Stachowiak and A.W. Batchelor. *Engineering Tribology*. 2006.

[88] Donald D. Heckelman and C. M. McC. Ettles. Viscous and inertial pressure effects at the inlet to a bearing film. *Tribology Transactions*, 31(1):1–5, 1988.

[89] W. Lewicki. Theory of hydrodynamic lubrication in parallel sliding. *The Engineer*, 200:939–941, 1955.

[90] M. Cz., K.W. Rodkiewicz, K.W. Kim, and J.S. Kennedy. On the significance of the inlet pressure build-up in the design of tilting-pad bearings. *Journal of Tribology*, 112: 17–22, 1990.

[91] E.J. Terrel, W.M. Needelman, and J.P. Kyle. *Wind Turbine Tribology in Green Tribology*. Springer, 2012.

[92] G.K. Nikas. A state-of-the-art review of the effects of particulate contamination and related topics in machine-element contacts. *Proceedings of the Institution of Mechanical Engineers, Part J: Journal of Engineering Tribology*, page 224, 2010.

[93] M.M. Khonsari and E.R. Booser. Effect of contamination on the performance of hydrodynamic bearings. *Proc. IMeche, Part J: J. Engineering Tribology*, 220(5):419–428, 2006.

[94] B.J. Roylance, J.A. Williams, and R. Dwyer-Joyce. Wear debris and associated wear phenomena - fundamental research and practice. *Proc. IMeche, Part J: J. Engineering Tribology*, 214(1):79–105, 2000.

[95] J.A. Willaims. Wear and wear particles - some fundamentals. *Tribol. Int.*, 38(10): 863–870, 2005.

[96] A. Sasaki. Contaminants in used oils and their problems. *Proc. IMeche, Part J: J. Engineering Tribology*, 220(5):471–478, 2006.

[97] M.R. Pavlat. Total cleanliness control for hydraulic and lubricating systems in the primary metal industry. *Lubr. Eng.*, 53(2):12–19, 1997.

[98] L. Badal, J. Whigham, and T. Minnick. The importance of iso cleanliness codes. *Machinery Lubrication Magazine*, 2005.

[99] M. Day. Clarifying the new iso contamination filtration standards. *Practicing Oil Analysis Magazine*, 2001.

[100] Z. Peng, N.J. Kessissoglou, and M. Cox. A study of the effect of contaminant particles in lubricants using wear debris and vibration condition monitoring techniques. *Wear*, 258(11-12):1651–1662, 2005.

[101] M.R. Sari, A. Haiahem, and L. Flamand. Effects of lubricant contamination on gear wear. *Tribol. Lett.*, 27(1):119–126, 2007.

[102] R.S. Sayles and E. Ioannides. Debris damage in rolling bearings and its effects on fatigue life. *Journal of Tribology*, 110(1):26–31, 1988.

[103] R.S. Sayles and P.B. Macpherson. *Influence of Wear Debris on Rolling Contact Fatigue in Rolling Contact Fatigue Testing of Bearing Steels*. ASTM STP, 1982.

[104] G.K. Nikas. Particle entrainment in elastohydrodynamic point contacts and related risks of oil starvation and surface indentation. *Journal of Tribology*, 124(3):461, 2002.

[105] S.C. Lin, T.C. Kuo, and C.C. Chieng. Particle trajectories around a flying slider. *Journal of Tribology*, 120(1):69–74, 1998.

[106] F. Dai and M.M. Khonsari. A continuum theory of a lubrication problem with solid particles. *Journal of Applied Mechanics*, 60(1):48, 1993.

[107] A. Kumar, S.R. Schmid, and W.R.D. Wilson. Particle behavior in two-phased lubrication. *Wear*, 206:130–135, 1997.

[108] P.G. Saffman. Lift on small sphere in slow shear flow. *J. Fluid Mech*, 22(2):385–400, 1965.

[109] A. Karnis, H.L. Goldsmith, and S.G. Mason. Flow of suspensions through tubes-5. *Chem. Eng*, 44(5):181–193, 1966.

[110] J.B. Hunt, A.J. Ryde-Weller, and F.A.H. Ashmead. Cavitation between meshing gear teeth. *Wear*, 71(1):65–78, 1981.

[111] X. He, H. Xiao, J.P. Kyle, E.J. Terrell, and H. Liang. Two-dimensional nanostructured y2o3 particles for viscosity modification. *Applied Physics Letters*, 104, 2014.

[112] L.R. Rudnick. *Lubricant Additives: Chemistry and Applications*. CRC Press, 2009.

[113] N.S. Ahmed and A.M. Nassar. *Lubricating Oil Additives*. InTech, 2011.

[114] M.J.S.d. Carvalho, P.R. Seidl, C.R.P. Belchior, and J.R. Sodré. Lubricant viscosity and viscosity improver additive effects on diesel fuel economy. *Tribology International*, 43(12):2298–2302, 2010.

[115] H.J. Herrmann, J.S. Andrade, A.D. Araújo, and M.P. Almeida. Particles in fluids. *The European Physical Journal Special Topics*, 143(181-189), 2007.

[116] L.G. Leal. Particle motions in a viscous fluid. *Annual Review of Fluid Mechanics*, 12 (436-476), 1980.

[117] E.Y.A. Wornyoh, V.K. Jasti, and C. F. Higgs III. A review of dry particulate lubrication: Powder and granular materials. *Journal of Tribology*, 129(2):438–449, 2007.

[118] S.M. Hsu. Nanolubrication: Concept and design. *Tribology International*, 37(7):537–545, 2004.

[119] V. Nicolosi, M. Chhowalla, M.G. Kanatzidis, M.S. Strano, and J.N. Coleman. Liquid exfoliation of layered materials. *Science*, 340(6139), 2013.

[120] T.W. Scharf and S.V. Prasad. Solid lubricants: A review. *Journal of Materials Science*, 48(2):511–531, 2013.

[121] L. Yadgarov, V. Petrone, R. Rosentsveig, Y. Feldman, R. Tenne, and A. Senatore. Tribological studies of rhenium doped fullerene-like mos2 nanoparticles in boundary, mixed, and elasto-hydrodynamic lubrication conditions. *Wear*, 297(1-2):1103–1110, 2013.

[122] V. Eswaraiah, V. Sankaranarayanan, and S. Ramaprabhu. Graphene-based engine oil nanofluids for tribological applications. *ACS Applied Materials Interfaces*, 3(11): 4221–4227, 2011.

[123] H.D. Huang, J.P. Tu, L.P. Gan, and C.Z. Li. An investigation on tribological properties of graphite nanosheets as oil additive. *Wear*, 261(2):140–144, 2006.

[124] X. He, Y. Chen, H. Zhao, H. Sun, X. Lu, and H. Liang. Y2o3 nanosheets as slurry abrasives for chemical-mechanical planarization of copper. *Friction*, 1(4):327–332, 2013.

[125] Jonathan P. Kyle and Elon J. Terrell. Application of smoothed particle hydrodynamics to full-film lubrication. *Journal of Tribology*, 135(4):041705–041705, 2013.

[126] X. He, Y. Zhou, and H. Liang. Cu-assisted synthesis of multi- and single-phase yttrium oxide nanosheets. *Journal of Materials Chemistry C*, 41(1):6829–6834, 2013.

[127] A.Z. Szeri. *Fluid Film Lubrication*. Cambridge University Press, 2010.

[128] H.A. Barnes. Thixotropy-a review. *Journal of Non-Newtonian Fluid Mechanics*, 70 (1-2):1–33, 1997.

[129] D.R. Foss and J.F. Brady. Structure, diffusion, and rheology of brownian suspensions by stokesian dynamics simulations. *J. Fluid Mech*, 407:167–200, 2000.

[130] P.J. Hoogerbrugge and J.M.V.A Koelman. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *Europhys Lett*, 19(155):96–160, 1992.

[131] E. Boek, P.V. Coveney, H.N. Lekkerkerker, and P. van der Schoot. Simulating the rheology of dense colloidal suspensions using dissipative particle dynamics simulations. *Phys Rev E*, 55:3124–3133, 1997.

[132] N.S. Martys. Study of a dissipative particle dynamics based approach for modeling suspensions. *J. Rheol.*, 49:401–424, 2005.

[133] N.S. Martys, W.L. George, B. Chun, and D. Lootens. A smoothed particle hydrodynamics-based fluid model with a spatially dependent viscosity: application to flow of a suspension with a non-newtonian fluid matrix. *Rheol Acta*, 49:1059–1069, 2010.

[134] J.J. Monaghan. Smoothed particle hydrodynamics. *Rep Prog Phys*, 68:1703–1759, 2005.

[135] P. Español and M. Revenga. Smoothed dissipative particle dynamics. *Phys Rev E*, 67, 2003.

[136] L.D. Landau and E.M. Lifshitz. *Fluid mechanics.* Pergamon, Oxford, 1987.

[137] H. Zhu, N.S. Martys, C. Ferraris, and D. DeKee. A numerical study of the flow of bingham-like fluids in two-dimensional vane and cylinder rheometers using a smoothed particle hydrodynamics (sph) based approach. *J. Non-Newton. Fluid Mech.*, 165:362–375, 2010.

[138] M.P. Allen and D.J. Tildesley. *Computer simulation of liquids.* Claredon, Oxford, 1987.

[139] N.S. Martys and R.D. Mountain. Velocity verlet algorithm for dissipative-particle-dynamics based models of suspension. *Phys Rev E*, 59(3733-3736), 1999.

[140] W.G. Hoover. Isomorphism linking smooth particles and embedded atoms. *Physica A*, 260:244, 1998.

[141] A. Colagrossi and M. Landrini. Numerical simulation of interfacial flows by smoothed particle hydrodynamics. *J. Comput. Phys*, 191(448), 2003.

[142] R. Clift, J.R. Grace, and M.E. Weber. *Bubbles Drops and Particles*. Academic Press, 1978.

[143] W.R. Jr. Jones and L.D. Wedeven. Surface-tension measurements in air of liquid lubricants to 200c by the differential-maximum-bubble-pressrure technique. *National Aeronautics and Space Administration Technical Note*, 1971.

[144] J.N. Israelachvili. *Intermolecular and Surface Forces*. Elsevier, 2011.

[145] J.P. Morris. Simulating surface tension with smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 33(3):333–353, 2000.

[146] S. Nugent and H.A. Posch. Liquid drops and surface tension with smoothed particle applied mechanics. *Physical Review E*, 62(4968), 2000.

[147] A.M. Tartakovsky and P. Meakin. Pore scale modeling of immiscible and miscible fluid flows using smoothed particle hydrodynamics. *Advances in Water Resources*, 29: 1464–1478, 2006.

[148] A.M. Tartakovsky and P. Meakin. Modeling of surface tension and contact angles with smoothed particle hydrodynamics. *Physical Review E*, 72.

[149] G. Gonnella, E. Orlandini, and J.M. Yeomans. Lattice boltzmann simulations of lamellar and droplet phases. *Phys. Rev. E*, 58(480), 1998.

[150] X. Shan and H. Chen. Simulation of nonideal gases and liquid-gas phase transitions by the lattice boltzmann equation. *Phys. Rev. E*, 47(1815), 1993.

[151] G. Triantafyllou, G.N. Angelopoulos, and P. Nikolopoulos. Surface and grain-boundary energies as well as surface mass transport in polycrystalline yttrium oxide. *J. Mater. Sci.*, 45(8):2015–2022, 2010.

# 7 Appendices

## 7.1 Nomenclature

$a$ acceleration associated with a body force, usually gravity 72

$\alpha$ directional index 6

$\beta$ directional index 6

$c$ artificial speed of sound of fluid used in equation of state 7, 8

$D$ problem dependent parameter used in calculating repulsive force 17

$d$ relative distance between two particles 11

$\delta$ Dirac delta function 9

$\delta^*$ percentage of density variation 8

$\epsilon$ shear strain rate tensor 6

$\eta$ dynamic viscosity of fluid 1, 6, 28

$\eta_s$ viscosity of suspending fluid 1

$F$ body force per unit mass 8

$f_a$ particle acceleration 24

$\gamma$ Surface energy/tension 72–74

$h$ film thickness height beneath pad 28

$h_1$ film thickness height at pad inlet 30

$h_2$ film thickness height at pad exit 31, 32

$\theta_c$ contact angle 81

$U$ pad sliding speed 28

$u_\alpha$ velocity component of fluid in the $\alpha$ direction 6

$V_b$ bulk fluid velocity 8

$V$ Computational domain size 83

$W$ SPH smoothing function 9

$W^*$ non-dimensional load carrying capacity 35

$W_1$ Word of adhesion/cohesion 73, 74

$x$ position vector 6

$\zeta$ constant for controlling the strength of the average velocity 20, 50

## 7.2 Artificial Surface Tension

Upon further investigation with the viscosity reduction seen in Section 4.4, it became apparent that this local drop in viscosity surrounding each NS was really attributable to a computational artifact stemming from the calculation of the density of the fluid (see Eqn. 19) around the immediate interface between the NS and the fluid. The issue is similar to a particle deficiency as seen in Fig. 3. The sharp discontinuity between the density of the NS and of the fluid leads to incorrect calculations of the density in the surrounding fluid. Hoover [140] and Colagrossi and Landrini [141] found that the standard SPH formulation of Gingold and Monaghan [33] creates artificial surface tension on the boundary between two fluids or a solid surface submerged in a fluid with different densities. This then provides a significant clue for the physical underlying mechanism for the non-Einstein-like viscosity drop seen in both experiment and computational simulations. For if an artificial surface tension could decrease the composite fluid viscosity, perhaps incorporating a real surface tension model into the simulations would do so as well, which is presented in Section 5.

## 7.3 Fortran Source Files

The following code can be compiled and run with most Fortran compilers (there are very few new language features implemented, so even Fortran 77 compilers should be able to run with without any errors). Most of the parameters can be changed in the param.inc file. The basic structure of this code was modeled after work by Liu [43].

### 7.3.1 param.inc

```
c-----------------------------------------------------------------------
c     Including file for parameters and constants used
c     in the entire SPH software packages.
c-----------------------------------------------------------------------

c     dim : Dimension of the problem (1, 2 or 3)
      integer dim
      parameter ( dim = 3)

c     maxn     : Maximum number of particles
c     max_interaction : Maximum number of interaction pairs
      integer maxn, max_interaction
      parameter ( maxn     = 500000,
     &            max_interaction = 15 * maxn )

c     Parameters for the computational geometry,
c     This defines the limits of the particle searching scheme.
c     Once a particle leaves these limits, the simulation
c     terminates...
c     x_maxgeom : Upper limit of allowed x-regime
c     x_mingeom : Lower limit of allowed x-regime
c     y_maxgeom : Upper limit of allowed y-regime
c     y_mingeom : Lower limit of allowed y-regime
c     z_maxgeom : Upper limit of allowed z-regime
c     z_mingeom : Lower limit of allowed z-regime
      double precision x_maxgeom, x_mingeom, y_maxgeom,
     &                 y_mingeom, z_maxgeom, z_mingeom
      parameter ( x_maxgeom =  2.1e-6,
     &            x_mingeom = 0     ,
```

```
     &             y_maxgeom =  195e-9   ,
     &             y_mingeom = 0,
     &             z_maxgeom =  2.1e-6      ,
     &             z_mingeom =  0      )

c     SPH algorithm for particle approximation (pa_sph)
c     pa_sph = 1 : (e.g. (p(i)+p(j))/(rho(i)*rho(j))
c              2 : (e.g. (p(i)/rho(i)**2+p(j)/rho(j)**2)
c              3 : DSPH
      integer pa_sph
      parameter(pa_sph = 3)

c     Put nanomaterials into fluid
c     nano = 1: nanoparticles
c            2: nanosheets
c            3: "freeze" SPH into nanoparticles
c            4: "freeze" SPH into nanosheets
      integer nano
      parameter(nano = 2)

c     Nearest neighbor particle searching (nnps) method
c     nnps = 1 : Simplest and direct searching
c            2 : Sorting grid linked list
      integer nnps
      parameter(nnps = 2 )

c     Smoothing length evolution (sle) algorithm
c     sle = 0 : Keep unchanged,
c           1 : h = fac * (m/rho)^(1/dim)
c           2 : dh/dt = (-1/dim)*(h/rho)*(drho/dt)
c           3 : Other approaches (e.g. h = h_0 * (rho_0/rho)**(1/
   dim) )

      integer sle
      parameter(sle = 0)

c     Smoothing kernel function
```

```
c     skf = 1, cubic spline kernel by W4 - Spline (Monaghan 1985)
c         = 2, Gauss kernel   (Gingold and Monaghan 1981)
c         = 3, Quintic kernel (Morris 1997)
c         = 4, use for SDPD calculations
      integer skf
      parameter(skf = 1)

c     Switches for different senarios

c     summation_density = .TRUE. : Use density summation model in
   the code,
c                         .FALSE.: Use continuiity equation
c     number_density = .TRUE. : Use number_density instead of rho
   to prevent
                  artificial surface tension. Note:
                     summation_density
                  still has to be on .TRUE. for this to work
                           .FALSE.: Regular density is used
c     average_velocity = .TRUE. : Monaghan treatment on average
   velocity,
c                        .FALSE.: No average treatment.
c     virtual_part = .TRUE. : Use virtual particle,
c                    .FALSE.: Dont use virtual particle.
c     ghost_part = .TRUE. : Use ghost particle,
c                  .FALSE.: Dont use ghost particle.
c     probe_part = .TRUE. : Use probe particles,
c                go from Langrangian to Eulerian
c                  .FALSE. No probe particles
c     visc = .TRUE. : Consider viscosity,
c           .FALSE.: No viscosity.
c     var_visc = .TRUE.: Have non-Newtonian viscosity
                       .FALSE.: Constant viscosity
c     ex_force =.TRUE. : Consider external force,
c               .FALSE.: No external force.
c     self_gravity = .TRUE. : Considering self_gravity,
c                    .FALSE.: No considering of self_gravity
c     nor_density =  .TRUE. : Density normalization by using CSPM,
```

```
c                    .FALSE.: No normalization.
c     froze = .TRUE. : freeze SPH particle for rigid body rotation
c             .FALSE.: simulate just fluid SPH particles
c     spart = .TRUE. : Calculate rigid body rotations
c                 .FALSE.: No rigid body calculations
c    output_rec = .TRUE.: Save sate of simulation
                            in case you want to restart
                            .FALSE.: Don't save state
c    surface_tens = .TRUE.: Turn surface tension on
                              .FALSE.: Turn surface tension off

      logical summation_density, average_velocity, virtual_part
     &         visc, ex_force, self_gravity, nor_density,
     &        probe_particle, ghost_particle, output_rec,
     &        spart, var_visc, froze, number_density,
     &        surface_tens

      parameter ( summation_density  = .true. )
      parameter ( number_density = .true.     )
      parameter ( average_velocity  = .true. )
      parameter ( virtual_part  = .false. )
      parameter ( ghost_particle = .false. )
      parameter ( probe_particle = .false. )
      parameter ( visc  = .true.  )
      parameter ( ex_force  = .false.)
      parameter ( self_gravity  = .false. )
      parameter ( nor_density  = .true. )
      parameter ( output_rec = .false. )
      parameter ( spart = .false. )
      parameter ( var_visc = .true.)
      parameter ( froze = .false.)
      parameter ( surface_tens = .false.)

c     Symmetry of the problem
c     nsym = 0 : no symmetry,
c          = 1 : axis symmetry,
c          = 2 : center symmetry.
```

```
      integer     nsym
      parameter ( nsym = 0)

c     Control parameters for output
c     int_stat = .true. : Print statistics about SPH particle
   interactions.
c                         including virtual particle information.
c     print_step: Print Timestep (On Screen)
c     save_step : Save Timestep    (To Disk File)
c     moni_particle: The particle number for information
   monitoring.
      logical int_stat
      parameter ( int_stat = .false. )
      integer print_step, save_step, moni_particle, rec_step
      parameter ( print_step = 100,
     &            save_step = 1000,
     &            moni_particle = 1600,
     &            rec_step = 10000  )

      double precision pi
c     shearcavity = .true. : carry out shear cavity simulation
      parameter ( pi = 3.14159265358979323846 )

c     Speed of sound for material
      double precision co
      parameter ( co = 4.26 )

c     Simulation cases
c     shocktube = .true. : carry out shock tube simulation
      logical shocktube, shearcavity, slidingwedge, waterdischarge
         ,
     &  gears
      parameter ( shocktube  = .false. )
      parameter ( shearcavity  = .true. )
      parameter ( slidingwedge = .false. )
      parameter ( waterdischarge = .false. )
      parameter ( gears = .false. )
```

```
c     sphon = .false. : shut-off SPH calculations and just perform
c                       Steady State D.E.M. sim with background
   particles
c                       having an itype = 10.
c     interon = .false. : do not perform linked list to determine
c                       interactions
      logical sphon,interon
      parameter ( sphon = .true. )
      parameter ( interon = .true. )
```

### 7.3.2 sdpd.f

```
      program SDPD

c
-----------------------------------------------------------------

c     This is a three dimensional SDPD code for simulating
   nanoparticle
c     additives and their effect on lubricant viscosity. the
   followings
c     are the basic parameters needed in this code or calculated
   by this
c     code.
c
c     Written by Jonathan Kyle 2013, Columbia University,
   Mechanical
c     Engineering Department, Energy & Tribology Laboratory.

c     mass-- mass of particles                                  [in]
c     ntotal-- total particle number ues                    [in]
c     dt-- Time step used in the time integration  [in]
c     itype-- types of particles                                 [in
   ]
c     x-- coordinates of particles                              [in/
   out]
```

```
c     vx— velocities of particles                                    [in
   /out]
c     rho— dnesities of particles                                    [in/
   out]
c     p— pressure  of particles                                      [in/
   out]
c     hsml— smoothing lengths of particles          [in/out]
c     c— sound velocity of particles                      [out]


      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: itype(:)
      integer ntotal, maxtimestep, d, m, i, yesorno,
     &        nstart,endgrab,nprobe,ierr,j,nspart,nrec,nplatelet,
     &        pm, k, nvirt,mp,np,op,nxp,nyp,nzp,l

      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
         (:),
     &     p(:),u(:),c(:),s(:),e(:),hsml(:),km(:),gm(:),von(:),dvx
        (:,:),
     &      Rt(:,:,:),Rx(:,:),Ry(:,:),Rz(:,:),Rf(:,:),Rtest(:,:),
     &     Pmom(:,:),Lmom(:,:),Ibody(:,:,:),invIbody(:,:,:),
        numDens(:)
      double precision  dt,dx,dy,dist
      double precision s1, s2,endtime,xl,yl,hx,hy,rcirc,xcirc,
         ycirc,r,
     &  theta,ap, pl, ph, pn, imomx, imomy, imomz, totalmass,gamma
         ,beta,
     &  alpha,imom,zl,dz,v_inf
      character arec*3
      integer*4 timeArray(3)
      real rand,xangleMax,xangleMin,zMin,zMax,xMin,xMax,yMin,yMax,
         xp,yp,
     &   zp,yangleMax,yangleMin,zangleMax,zangleMin
```

```
c ---- The Interface section allows the capability to pass
   allocatable ----
c      arrays to subroutines
       INTERFACE
          SUBROUTINE input(x,vx,mass,rho,p,u,itype,hsml,ntotal,xl,
              yl,km,
     &                        gm,von,nspart,totalmass,pm,Rt,Lmom,Pmom,
         Ibody,
     &                     invIbody,nplatelet,zl,numDens)
          REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),
              rho(:),
     &                       p(:), u(:), hsml(:), km(:), gm(:), von(:)
         ,
     &                     Rt(:,:,:),Lmom(:,:),Pmom(:,:),Ibody(:,:,:)
         ,
     &                     invIbody(:,:,:),numDens(:)
          INTEGER, ALLOCATABLE :: itype(:)
          integer :: ntotal,nspart,pm,nplatelet
          double precision :: xl,yl,pl,ph,pn,totalmass,zl
          END SUBROUTINE input

          SUBROUTINE output_virt(x,vx,rho,p,mass,itype,nvirt,nspart)
          REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),rho
              (:),
     &                    p(:)
          INTEGER, ALLOCATABLE :: itype(:)
          integer i
          INTEGER :: nvirt, nspart
          END SUBROUTINE output_virt

          SUBROUTINE time_integration(x,vx,mass,rho,p,u,c,s,e,itype
              ,hsml,
     &         ntotal,maxtimestep,dt,nstart,endtime,endgrab,xl,yl,km
         ,gm,
     &         von,dvx,nspart,ap,imom,totalmass,nplatelet,pm,Rt,Lmom
         ,Pmom,
     &         Ibody,invIbody,nvirt,zl,numDens)
```

```
      REAL (KIND=8), ALLOCATABLE :: x(:,:),vx(:,:),mass(:),rho
         (:),
     &    p(:),u(:),c(:),s(:),e(:),hsml(:),km(:),gm(:),von(:),dvx
        (:,:),
     &    Rt(:,:,:),Lmom(:,:),Pmom(:,:),Ibody(:,:,:),invIbody
        (:,:,:),
     &    numDens(:)
      INTEGER, ALLOCATABLE :: itype(:)
      integer :: ntotal,maxtimestep, nstart,endgrab,nspart,
         nplatelet,
     &           pm,nvirt
      double precision :: dt, endtime, xl, yl,ap,imom,totalmass
         ,zl
      END SUBROUTINE time_integration
    END INTERFACE
c
-------------------------------------------------------------------------------


c *** By using allocatable arrays, we can reduce the filesize of
   the program
c *** so that Windows can run it. Use the ALLOCATE statement to
   define
c *** the size of the allocatable arrays at runtime

      ALLOCATE (x(1:3,1:maxn),vx(1:3,1:maxn),mass(maxn),rho(maxn)
         ,
     &    p(maxn),u(maxn),hsml(maxn),itype(maxn),km(maxn),
     &    gm(maxn),von(maxn),dvx(1:3,1:maxn),Rt(1:3,1:3,1:1),
     &    Rx(1:3,1:3),Ry(1:3,1:3),Rz(1:3,1:3),Rf(1:3,1:3),
     &    Rtest(1:3,1:3),Pmom(1:3,1:1),Lmom(1:3,1:1),
     &    Ibody(1:3,1:3,1:1),invIbody(1:3,1:3,1:1),numDens(maxn)
        ,
     &    STAT=IERR)
```

```
c ---- Print the current date & time to the command line
      call time_print

c ---- Determine time step size depending on the input routine
   -----
!!!       if (shearcavity) dt = 1e-13
!         if (shearcavity) dt = 0.6e-13
          if (shearcavity) dt = 0.2e-13
!!         if (shearcavity) dt = 2e-15
c        if (shearcavity) dt = 1e-10

c ---- Initialize time step and output stamp intervals
      nstart = 0
      endtime = 0
      endgrab = 0

c ---- Call input method to determine intial variables and geometry
    ---
 2     write(*,*)'Run from beginning of simulation or recovery
    file ?'
      write(*,*)'***** (0=Beginning, 1=Recovery File)
         ************'
      read(*,*) yesorno

      if (yesorno.eq.0) then
        nspart = 0
        ntotal = 0
       call input(x,vx,mass,rho, p, u, itype, hsml, ntotal,xl,yl,
          km,gm,
     &           von,nspart,totalmass,pm,Rt,Lmom,Pmom,Ibody,
         invIbody,
     &           nplatelet,zl,numDens)


      elseif (yesorno.eq.1) then
       if (shearcavity) then
        write(*,*)'Recovery File Number:'
```

```
      read(*,*) nrec
      write(arec,'(i3.3)') nrec
      open(59,file='Recovery_file_'//arec//'.txt')
      read(59,*) nstart
      read(59,*) endtime
      read(59,*) endgrab
      read(59,*) ntotal
      read(59,*) nspart
      read(59,*) nvirt
      do i=1,ntotal+nspart+nvirt
        read(59,*) x(1,i),x(2,i),x(3,i)
      enddo
      do i=1,ntotal+nspart+nvirt
        read(59,*) vx(1,i),vx(2,i),vx(3,i)
      enddo
      do i=1,ntotal+nspart+nvirt
        read(59,*) rho(i)
      enddo
      do i=1,ntotal+nspart+nvirt
        read(59,*) mass(i)
      enddo
      do i=1,ntotal+nspart+nvirt
        read(59,*) p(i)
      enddo
      do i=1,ntotal+nspart+nvirt
        read(59,*) itype(i)
      enddo
      do i=1,ntotal+nspart
        read(59,*) hsml(i)
      enddo
      do i=1,ntotal+nspart
        read(59,*) dvx(1,i), dvx(2,i), dvx(3,i)
      enddo
    endif


else
```

```
        write(*,*) '!!!! Please enter either 1 or 0 !!!!!!'
        go to 2
       endif

 1     write(*,*)'
     ************************************************'
       write(*,*)'          Please input the number of time steps
          '
       write(*,*)'
          ************************************************'
       read(*,*) maxtimestep
c ---- Call the main time integration method. This is the main
     program ----
       call time_integration(x, vx, mass, rho, p, u, c, s, e, itype
          ,
     &      hsml, ntotal, maxtimestep, dt,nstart,endtime,endgrab,xl
         ,yl,
     &      km,gm,von,dvx,nspart,ap,imom,totalmass,nplatelet,pm,Rt,
        Lmom,
     &      Pmom,Ibody,invIbody,nvirt,zl,numDens)

        write(*,*)'
          ************************************************'
        write(*,*) 'Are you going to run more time steps ? (0=No, 1=
          yes)'
        write(*,*)'
          ************************************************'
        read (*,*) yesorno
        if (yesorno.ne.0) go to 1
        call time_print


        end
```

### 7.3.3 av_vel.f

```
        subroutine av_vel(ntotal,mass,niac,pair_i,pair_j,
     &            w, vx, rho, av,itype,imark,x,hsml,numDens)
```

```
c
c-----------------------------------------------------------------------

c     Subroutine to calculate the average velocity to correct
   velocity
c     for preventing penetration (monaghan, 1992) and generating
c     no-slip boundary conditions at interfaces.

c     ntotal : Number of particles                                  [in]
c     mass   : Particle masses                                      [in]
c     niac   : Number of interaction pairs                        [in]
c     pair_i : List of first partner of interaction pair          [in]
c     pair_j : List of second partner of interaction pair         [in]
c     w      : Kernel for all interaction pairs                     [in]
c     vx     : Velocity of each particle                             [in
   ]
c     rho    : Density of each particle                             [in]
c     av     : Average velocityof each particle                  [out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: itype(:), pair_i(:),pair_j(:),
         solidinc(:),
     &        imark(:)
      integer ntotal, niac, IERR
      REAL (KIND=8), ALLOCATABLE :: mass(:), w(:), vx(:,:), rho(:)
         ,
     &        av(:,:),x(:,:),hsml(:), numDens(:)
      REAL (KIND=8) :: dvpx(1:3),dx(1:3),dwdx(1:3)
      integer i,j,k,d,start
      double precision   vcc, epsilon,r,aw,vxf, xl, yl, zl

      INTERFACE
          SUBROUTINE kernel(r,dx,hsml,aw,dwdx)
          double precision r, hsml, aw
```

```
          REAL (KIND=8) :: dwdx(1:3),dx(1:3)
          END SUBROUTINE
        END INTERFACE

       ALLOCATE (solidinc(1:maxn))


c      epsilon ---- a small constants chosen by experience, may lead
c      to instability.for example, for the 1 dimensional shock tube
c      problem, the E <= 0.3*** I believe Antoci uses
c       (1-theta) = epsilon where theta = .92 for water discharge
c     problem. i.e. his epsilon is a small .08

       epsilon = 0.3

c$OMP PARALLEL
c$OMP do
       do i = 1, ntotal
         do d = 1, dim
           av(d,i) = 0.
         enddo
       enddo
c$OMP enddo
c$OMP END PARALLEL


c$OMP PARALLEL
c$OMP do PRIVATE(i,j,dvpx)
       do k=1,niac
         i = pair_i(k)
         j = pair_j(k)
         if(itype(i).ne.4.and.itype(j).ne.4) then
          if((itype(i)+itype(j)).ge.0) then
           do d=1,dim
              dvpx(d) = (vx(d,i) - vx(d,j))
              if(number_density) then
```

```
                av(d, i) = av(d,i) - 2*mass(j)*dvpx(d)/((numDens(i)
                   *
     &                       mass(i))+(numDens(j)*mass(j)))*w(k)
                av(d, j) = av(d,j) + 2*mass(i)*dvpx(d)/((numDens(i)
                   *
     &                       mass(i))+(numDens(j)*mass(j)))*w(k)
              else
              av(d, i) = av(d,i) - 2*mass(j)*dvpx(d)/(rho(i)+rho(j)
                 )*w(k)
              av(d, j) = av(d,j) + 2*mass(i)*dvpx(d)/(rho(i)+rho(j)
                 )*w(k)
              endif
            enddo
          endif
        endif
      enddo
c$OMP enddo
c$OMP END PARALLEL



c$OMP PARALLEL
c$OMP do
      do i=1,ntotal
        if(itype(i).eq.2) then
          do d=1, dim
            av(d,i) = epsilon * av(d,i)
          enddo
        endif
      enddo
c$OMP enddo
c$OMP END PARALLEL



      end
```

### 7.3.4 density.f

```
      subroutine sum_density(ntotal,hsml,mass,niac,pair_i,pair_j,w
     ,
     &           itype,rho,numDens)

C
     ------------------------------------------------------------------

C  Subroutine to calculate the density with SPH summation
   algorithm.

C     ntotal : Number of particles                    [in]
C     hsml   : Smoothing Length                     [in]
C     mass   : Particle masses                        [in]
C     niac   : Number of interaction pairs          [in]
C     pair_i : List of first partner of interaction pair  [in]
C     pair_j : List of second partner of interaction pair [in]
C     w      : Kernel for all interaction pairs     [in]
c     itype   : type of particles                          [in]
c     x       : Coordinates of all particles         [in]
c     rho     : Density                                    [
   out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:), itype(:),
         imark(:)
      REAL (KIND=8), ALLOCATABLE :: hsml(:), mass(:), w(:), rho(:)
     ,
     &           wi(:), psitype(:), numDens(:)
      REAL (KIND=8) :: hvs(1:3),hv(1:3)
      integer ntotal, niac, IERR
      double precision dcolor
      integer i, j, k, d
      double precision selfdens, r
```

```
      INTERFACE
         SUBROUTINE kernel(r,hv,hsml,selfdens,hvs)
         double precision r, hsml, selfdens
         REAL (KIND=8) :: hv(1:3),hvs(1:3)
         END SUBROUTINE
      END INTERFACE


      ALLOCATE (wi(maxn),
     &     STAT=IERR)

c     wi(maxn)---integration of the kernel itself

      do d=1,dim
        hv(d) = 0.e0
        hvs(d) = 0.e0
      enddo

c     Self density of each particle: Wii (Kernel for distance 0)
c     and take contribution of particle itself:

      r=0.

c     Firstly calculate the integration of the kernel over the
   space

      if(nor_density) then
c$OMP PARALLEL Private(selfdens)
c$OMP do
      do i=1,ntotal
        if(abs(itype(i)).eq.2) then
          call kernel(r,hv,hsml(i),selfdens,hvs)
          wi(i)=selfdens*mass(i)/rho(i)
        endif
      enddo
c$OMP enddo
```

```
c$OMP END PARALLEL


c$OMP PARALLEL
c$OMP do Private(i,j,k)
      do k=1,niac
        i = pair_i(k)
        j = pair_j(k)
        if(abs(itype(i)).eq.abs(itype(j))) then
          wi(i) = wi(i) + (mass(j)/rho(j))*w(k)
          wi(j) = wi(j) + (mass(i)/rho(i))*w(k)
        endif
      enddo
c$OMP enddo
c$OMP End PARALLEL
      endif

c     Secondly calculate the rho integration over the space
c$OMP PARALLEL
c$OMP do PRIVATE(selfdens)
      do i=1,ntotal
        if(itype(i).le.2) then
          call kernel(r,hv,hsml(i),selfdens,hvs)
          rho(i) = selfdens*mass(i)
        endif
      enddo
c$OMP enddo
c$OMP END PARALLEL

c    If number_density, first calculate numDens over space
       if(number_density) then
c$OMP PARALLEL
c$OMP do PRIVATE(selfdens)
       do i = 1, ntotal
          call kernel(r,hv,hsml(i),selfdens,hvs)
         numDens(i) = selfdens
       enddo
```

```
c$OMP enddo
c$OMP END PARALLEL

c *** Calculate number density

c$OMP PARALLEL
c$OMP do PRIVATE(i,j)
        do k=1,niac
          i = pair_i(k)
          j = pair_j(k)
          numDens(i) = numDens(i) + w(k)
          numDens(j) = numDens(j) + w(k)
        enddo
c$OMP enddo
c$OMP END PARALLEL
        endif

c$OMP PARALLEL
c$OMP do PRIVATE(i,j)
       do k=1,niac
         i = pair_i(k)
         j = pair_j(k)
         if(nor_density) then
          if(abs(itype(i)).eq.abs(itype(j))) then
             rho(i) = rho(i) + mass(j)*w(k)
             rho(j) = rho(j) + mass(i)*w(k)
          endif

         else
             rho(i) = rho(i) + mass(j)*w(k)
             rho(j) = rho(j) + mass(i)*w(k)
         endif
       enddo
c$OMP enddo
c$OMP END PARALLEL

c      Thirdly, calculate the normalized rho, rho=sum(rho)/sum(w)
```

```
      if (nor_density) then
c$OMP PARALLEL
c$OMP do
        do i=1, ntotal
          if(abs(itype(i)).eq.2) then
             rho(i)=rho(i)/wi(i)
          endif
        enddo
c$OMP enddo
c$OMP END PARALLEL
      endif

c     Replace density of solid particles with their value

c$OMP PARALLEL
c$OMP do
        do i = 1, ntotal
         if(itype(i).eq.11.or.itype(i).eq.12) then
            rho(i) = 5010
         endif
        enddo
c$OMP enddo
c$OMP END PARALLEL




      end

      subroutine con_density(ntotal,mass,niac,pair_i,pair_j,
     &           dwdx,vx,itype,drhodt,imark)

c
```

```
c     Subroutine to calculate the density with SPH continuiity
   approach.

c     ntotal : Number of particles                [in]
c     mass   : Particle masses                    [in]
c     niac   : Number of interaction pairs    [in]
c     pair_i : List of first partner of interaction pair    [in]
c     pair_j : List of second partner of interaction pair [in]
c     dwdx   : derivation of Kernel for all interaction pairs  [in
   ]
c     vx     : Velocities of all particles        [in]
c     itype   : type of particles                   [in]
c     x      : Coordinates of all particles      [in]
c     rho    : Density                                  [in]
c     drhodt : Density change rate of each particle  [out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:), itype(:),
         imark(:)
      integer ntotal,niac
      REAL (KIND=8), ALLOCATABLE :: mass(:), dwdx(:,:), vx(:,:), x
         (:,:),
     &       rho(:), drhodt(:)
      integer i,j,k,d, IERR
      double precision  vcc, dvx(3)

      ALLOCATE (imark(1:maxn), STAT=IERR)

c$OMP PARALLEL
c$OMP do
      do i = 1, ntotal
        drhodt(i) = 0.
      enddo
c$OMP enddo
c$OMP END PARALLEL
```

```
c$OMP Parallel
c$OMP Do Private(i,j,dvx,vcc)
      do k=1,niac
        i = pair_i(k)
        j = pair_j(k)
        do d=1,dim
          dvx(d) = vx(d,i) - vx(d,j)
        enddo
        vcc = dvx(1)*dwdx(1,k)
        do d=2,dim
          vcc = vcc + dvx(d)*dwdx(d,k)
        enddo

        if(abs(itype(i)).eq.2.and.abs(itype(j)).eq.2) then
c$OMP FLUSH(drhodt)
c$ATOMIC
           drhodt(i) = drhodt(i) + mass(j)*vcc
c$OMP FLUSH(drhodt)
c$ATOMIC
           drhodt(j) = drhodt(j) + mass(i)*vcc
        endif

      enddo
c$OMP Enddo
c$OMP End Parallel

      end
```

### 7.3.5 direct_find.f

```
      subroutine direct_find(itimestep, ntotal,hsml,x,niac,pair_i,
     &           pair_j,w,dwdx,countiac)

c
```

---

```
c   Subroutine to calculate the smoothing funciton for each
   particle and
c   the interaction parameters used by the SPH algorithm.
   Interaction
c   pairs are determined by directly comparing the particle
   distance
c   with the corresponding smoothing length.

c     itimestep : Current time step                [in]
c     ntotal    : Number of particles              [in]
c     hsml      : Smoothing Length                 [in]
c     x         : Coordinates of all particles     [in]
c     niac      : Number of interaction pairs      [out]
c     pair_i    : List of first partner of interaction pair [out]
c     pair_j    : List of second partner of interaction pair [out]
c     w         : Kernel for all interaction pairs   [out]
c     dwdx    : Derivative of kernel with respect to x,y and z  [
   out]
c     countiac  : Number of neighboring particles  [out]

      implicit none
      include 'param.inc'

      integer itimestep, ntotal,niac,pair_i(max_interaction),
     &        pair_j(max_interaction), countiac(maxn)
      double precision hsml(maxn), x(3,maxn), w(max_interaction),
     &       dwdx(3,max_interaction)
      integer i, j, d,  sumiac, maxiac, miniac, noiac,
     &        maxp, minp, scale_k
      double precision dxiac(3), driac, r, mhsml, tdwdx(3)

      if (skf.eq.1) then
        scale_k = 2
      else if (skf.eq.2) then
        scale_k = 3
      else if (skf.eq.3) then
        scale_k = 3
```

```
      endif

      do i=1,ntotal
        countiac(i) = 0
      enddo

      niac = 0
c$OMP Parallel
c$OMP Do Private(dxiac,driac,mhsml,r,tdwdx)
      do i=1,ntotal-1
        do j = i+1, ntotal
          dxiac(1) = x(1,i) - x(1,j)
          driac    = dxiac(1)*dxiac(1)
          do d=2,dim
            dxiac(d) = x(d,i) - x(d,j)
            driac    = driac + dxiac(d)*dxiac(d)
          enddo
          mhsml = (hsml(i)+hsml(j))/2.
          if (sqrt(driac).lt.scale_k*mhsml) then
            if (niac.lt.max_interaction) then

c       Neighboring pair list, and totalinteraction number and
c       the interaction number for each particle

              niac = niac + 1
              pair_i(niac) = i
              pair_j(niac) = j
              r = sqrt(driac)
              countiac(i) = countiac(i) + 1
              countiac(j) = countiac(j) + 1

c       Kernel and derivations of kernel

              call kernel(r,dxiac,mhsml,w(niac),tdwdx)
              do d=1,dim
                dwdx(d,niac) = tdwdx(d)
              enddo
```

```
            else
              print *,
     &         ' >>> ERROR <<< : Too many interactions'
              stop
            endif
          endif
        enddo
      enddo
c$OMP Enddo
c$OMP End Parallel

c     Statistics for the interaction

      sumiac = 0
      maxiac = 0
      miniac = 1000
      noiac  = 0
      do i=1,ntotal
        sumiac = sumiac + countiac(i)
        if (countiac(i).gt.maxiac) then
          maxiac = countiac(i)
          maxp = i
        endif
        if (countiac(i).lt.miniac) then
          miniac = countiac(i)
          minp = i
        endif
        if (countiac(i).eq.0)       noiac  = noiac + 1
      enddo

      if (mod(itimestep,print_step).eq.0) then
        if (int_stat) then
          print *,' >> Statistics: interactions per particle:'
          print *,'**** Particle:',maxp, ' maximal interactions:',
             maxiac
          print *,'**** Particle:',minp, ' minimal interactions:',
             miniac
```

```
          print *,'**** Average :',real(sumiac)/real(ntotal)
          print *,'**** Total pairs : ',niac
          print *,'**** Particles with no interactions:',noiac
        endif
      endif

      end
```

### 7.3.6 eos.f

```
      subroutine p_gas(rho, u, p, c)

c
-----------------------------------------------------------------------

c   Gamma law EOS: subroutine to calculate the pressure and sound

c     rho     : Density                      [in]
c     u       : Internal energy        [in]
c     p       : Pressure             [out]
c     c       : sound velocity     [out]

      implicit none
      double precision rho, u, p, c
      double precision gamma

c      For air (idea gas)

      gamma=1.4
      p = (gamma-1) * rho * u
      c = sqrt((gamma-1) * u)

      end

      subroutine p_art_water(rho, p, rhonot, numDens, mass)

c
-----------------------------------------------------------------------
```

```
c   Artificial equation of state for the artificial
   compressibility

c     rho     : Density              [in]
c     u       : Internal energy      [in]
c     p       : Pressure            [out]
c     c       : sound velocity      [out]
c     Equation of state for artificial compressibility

      implicit none
      include 'param.inc'

      REAL (KIND=8), ALLOCATABLE :: vx(:,:)
      double precision rho, u, p, c, b, vxx, vyy, vmag, numDens
      double precision gamma, rhonot, eta, length, epsilon,mass
      integer i, itype

c     Artificial EOS, Form 1 (Monaghan, 1994)
c     gamma ususally equals 7 except in low reynolds
c     situations where it should equal 1
c      gamma=7.
c      rho0=1000
c      b = 392.4
c      p = b*((rho/rho0)**gamma-1)
c      c = 1480.

c     Artificial EOS, Form 2 (Morris, 1997)


      c = 1.18

      if(number_density) then
        p = (10 * numDens * ((2.05268e-6/163)**3)) / (rhonot)
      else
        p = c**2 * (rho - rhonot)
      endif
```

```
      end

      subroutine p_solid(rho, p, km)

c
     ------------------------------------------------------------------

c       Equation of State of Solids based on Bulk Modulus

c     rho     : Density          [in]
c     km      : Bulk Modulus     [out]
c     p       : Pressure             [out]
c     c       : sound velocity     [out]


        implicit none
        double precision rho,rho0, p, c, km

        rho0 = 1100
        c = sqrt(km/rho0)
        p = c**2 * (rho - rho0)

        end
```

### 7.3.7 external_force.f

```
      subroutine ext_force(ntotal,mass,x,niac,pair_i,pair_j,
     &           itype,hsml,dvxdt,numDens,dwdx,p,itimestep)

c
     ------------------------------------------------------------------

c     Subroutine to calculate the external forces, e.g.
   gravitational forces.
c     The forces from the interactions with boundary virtual
   particles
c     as well as surface tension,  is also calculated here.
```

```
c     ntotal  : Number of particles          [in]
c     mass    : Particle masses              [in]
c     x       : Coordinates of all particles     [in]
c     pair_i : List of first partner of interaction pair      [in]
c     pair_j : List of second partner of interaction pair    [in]
c     itype   : type of particles      [in]
c     hsml    : Smoothing Length    [in]
c     dvxdt   : Acceleration with respect to x, y and z     [out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: itype(:), pair_i(:), pair_j(:)
      integer ntotal, niac, IERR
      REAL (KIND=8), ALLOCATABLE :: mass(:), x(:,:), hsml(:),
     &     dvxdt(:,:),dx(:),numDens(:),dwdx(:,:),p(:)
      integer i, j, k, d, itimestep
      double precision  rr, f, rr0, dd, p1, p2 , s11, s12,
     &     xcent,ycent,zcent,xl,yl,zl,dz,dist_one,
     &     dist_two,pl,fij(1:3,1),po,pliq,vo,vl,dy


      xl = 2.05268e-6
      yl = 150e-9
      zl = 2.05268e-6

c      dx = xl/163
      dy = yl/12
      dz = zl/163

      do i = 1, ntotal
       do d = 1, dim
         dvxdt(d, i) = 0.
            enddo
      enddo
```

```
c      Consider self-gravity or not ?

       if (self_gravity) then
        do i = 1, ntotal
          dvxdt(dim, i) = -9.8
        enddo
       endif

c      s11: magnitude of surface tension amongst like particles
c      s12: magnitude of surface tension amongst unlike particles
       s11 = 5e-5
       s12 = 1e-10

c      Boundary particle force and penalty anti-penetration force.
c      dd problem dependent parameter: same scale as square of
   largest
c      velocity. rr0 is cutoff distance: close to initial particle
   spacing

       rr0 = 150e-9/12
       dd=.018

       p1 = 12
       p2 = 4
c$OMP PARALLEL
c$OMP do PRIVATE(i,j,rr,dx,f)
       do  k=1,niac
         ALLOCATE(dx(1:3))
         i = pair_i(k)
         j = pair_j(k)
         if(itype(i).eq.2.and.itype(j).lt.0.or.itype(i).lt.0.and.
            itype(j)
     &        .eq.2) then
           rr = 0.
           do d=1,dim
             dx(d) =  x(d,i) -  x(d,j)
             rr = rr + dx(d)*dx(d)
```

```
            enddo
            rr = sqrt(rr)
            if(rr.lt.rr0) then
              f = ((rr0/rr)**p1-(rr0/rr)**p2)/rr**2
              do d = 1, dim
c$OMP FLUSH(dvxdt)
c$OMP ATOMIC
                dvxdt(d, i) = dvxdt(d, i) + dd*dx(d)*f
              enddo
            endif
          endif
c *** Consider if surface tension is being used
          if (surface_tens) then
            rr = 0.
            do d = 1,dim
              dx(d) = x(d,i) - x(d,j)
              rr = rr + dx(d)*dx(d)
            enddo
              rr = sqrt(rr)
              if(rr.lt.(2*rr0)) then
                if(itype(i).eq.2.and.itype(j).eq.2) then
                  f = s11*cos((1.5*pi*rr)/(3*2*rr0))/rr
                    do d=1,dim
c$OMP FLUSH(dvxdt)
c$OMP ATOMIC
                      dvxdt(d, i) = dvxdt(d, i) + (f*dx(d))/mass(i)
c$OMP FLUSH(dvxdt)
c$OMP ATOMIC
                      dvxdt(d, j) = dvxdt(d, j) - (f*dx(d))/mass(j)
                    enddo
                  elseif(itype(i).eq.11.and.itype(j).eq.2.or.
     &                 itype(j).eq.11.and.itype(i).eq.2.or.itype(i).
     &                 eq.12.and.itype(j).eq.2.or.itype(j).eq.12.and.
     &                 itype(i).eq.2) then
                    f = s12*cos((1.5*pi*rr)/(3*2*rr0))/rr
                    do d=1,dim
c$OMP FLUSH(dvxdt)
```

```
c$OMP ATOMIC
                        dvxdt(d, i) = dvxdt(d, i) + (f*dx(d))/mass(i)
c$OMP FLUSH(dvxdt)
c$OMP ATOMIC
                        dvxdt(d, j) = dvxdt(d, j) - (f*dx(d))/mass(j)
                        enddo
                     endif
                    endif
                 endif
          DEALLOCATE(dx)
        enddo
c$OMP enddo
c$OMP END PARALLEL


        end
```

### 7.3.8 grid_geom.f

```
        subroutine grid_geom(i,xs,ngridx,maxgridx,mingridx,dgeomx,
           xgcell)

c
   ------------------------------------------------------------------


c    Subroutine to calculate the coordinates (xgcell) of the cell
   of
c    the sorting  grid, in which the particle with coordinates (x)
   lies.

c      x         : Coordinates of particle  [in]
c      ngridx: Number of sorting grid cells in x, y, z-direction  [
   in]
c      maxgridx : Maximum x-, y- and z-coordinate of grid range  [
   in]
c      mingridx : Minimum x-, y- and z-coordinate of grid range [in
   ]
c      dgeomx    : x-, y- and z-expansion of grid range   [in]
```

```
c      xgcell   : x-, y- and z-coordinte of sorting grid cell [out]

       implicit none
       include 'param.inc'

       REAL (KIND=8) :: xs(1:3),dgeomx(1:3),maxgridx(1:3),mingridx
          (1:3)
       INTEGER :: NGR,ngridx(1:3),xgcell(1:3)
       integer i,d


       do d=1,3
         xgcell(d) = 1
       enddo

       do d=1,dim
         if ((xs(d).gt.maxgridx(d)).or.(xs(d).lt.mingridx(d))) then
           print *,' >>> ERROR <<< : Particle out of range'
           print *,'     Particle position: x(',i,d,') = ',xs(d)
           print *,'     Range: [xmin,xmax](',D,') =
      &         [',mingridx(d),',',maxgridx(d),']'
           stop
         else
           xgcell(d) = int(real(ngridx(d))/dgeomx(d)*
      &         (xs(d)-mingridx(d)) + 1.e0)
         endif
       enddo

       end
```

### 7.3.9 hsml.f

```
       subroutine h_upgrade(dt,ntotal, mass, vx, rho, niac,
      &           pair_i, pair_j, dwdx, hsml)

c
```

---

```
c     Subroutine to evolve smoothing length

c     dt     : time step    [in]
c     ntotal : Number of particles  [in]
c     mass   : Particle masses      [in]
c     vx     : Velocities of all particles  [in]
c     rho    : Density      [in]
c     niac   : Number of interaction pairs [in]
c     pair_i : List of first partner of interaction pair [in]
c     pair_j : List of second partner of interaction pair [in]
c     dwdx : Derivative of kernel with respect to x, y and z [in]
c     hsml   : Smoothing Length     [in/out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: pair_i(:),pair_j(:)
      integer ntotal, niac
      REAL (KIND=8), ALLOCATABLE :: mass(:), vx(:,:), rho(:), dwdx
     (:,:),
     &       hsml(:), dvx(:), vcc(:), dhsml(:)
      integer i,j,k,d
      double precision dt, fac, hvcc

      if (sle.eq.0 ) then

c---  Keep smoothing length unchanged.

        return

      else if (sle.eq.2) then

c---  dh/dt = (-1/dim)*(h/rho)*(drho/dt).

        do i=1,ntotal
          vcc(i) = 0.e0
        enddo
```

```
      do k=1,niac
        i = pair_i(k)
        j = pair_j(k)
        do d=1,dim
          dvx(d) = vx(d,j) - vx(d,i)
        enddo
        hvcc = dvx(1)*dwdx(1,k)
        do d=2,dim
          hvcc = hvcc + dvx(d)*dwdx(d,k)
        enddo
        vcc(i) = vcc(i) + mass(j)*hvcc/rho(j)
        vcc(j) = vcc(j) + mass(i)*hvcc/rho(i)
      enddo

      do i = 1, ntotal
        dhsml(i) = (hsml(i)/dim)*vcc(i)
        hsml(i) = hsml(i) + dt*dhsml(i)
        if (hsml(i).le.0) hsml(i) = hsml(i) - dt*dhsml(i)
      enddo

    else if(sle.eq.1) then

      fac = 2.0
      do i = 1, ntotal
        hsml(i) = fac * (mass(i)/rho(i))**(1./dim)
      enddo

    endif

    end
```

### 7.3.10 init_grid.f

```
      subroutine init_grid(ntotal,hsml,grid,ngridx,ghsmlx,
     &           maxgridx,mingridx,dgeomx)
```

```
c
c-----------------------------------------------------------------------

c   Subroutine to established a pair linked list by sorting grid
    cell.
c   It is suitable for a homogeneous particle distribution with
    the
c   same smoothing length in an instant. A fixed number of
    particles
c   lie in each cell.

c      ntotal    : Number of particles  [in]
c      hsml      : Smoothing Length  [in]
c      grid      : array of grid cells   [out]
c      ngridx    : Number of sorting grid cells in x, y, z-direction
     [out]
c      ghsmlx    : Smoothing length measured in cells of the grid [
    out]
c      maxgridx  : Maximum x-, y- and z-coordinate of grid range [
    out]
c      mingridx  : Minimum x-, y- and z-coordinate of grid range[out
    ]
c      dgeomx    : x-, y- and z-expansion of grid range  [out]

       implicit none
       include 'param.inc'

c      Parameter used for sorting grid cells in the link list
    algorithm
c      maxngx   : Maximum number of sorting grid cells in x-
    direction
c      maxngy   : Maximum number of sorting grid cells in y-
    direction
c      maxngz   : Maximum number of sorting grid cells in z-
    direction
c      Determining maximum number of sorting grid cells:
c      (For an homogeneous particle distribution:)
```

```
c      1-dim. problem: maxngx = maxn ,  maxngy = maxngz = 1
c      2-dim. problem: maxngx = maxngy ~ sqrt(maxn) ,  maxngz = 1
c      3-dim. problem: maxngx = maxngy = maxngz ~ maxn^(1/3)
       integer maxngx,maxngy,maxngz,IERR
       parameter ( maxngx  = 160      ,
      &            maxngy  = 16       ,
      &            maxngz  = 160        )
       INTEGER, ALLOCATABLE :: grid(:,:,:), ghsmlx(:),
      &        maxng(:), ngrid(:)
       integer ntotal
       REAL (KIND=8):: maxgridx(1:3), mingridx(1:3), dgeomx(1:3)
       INTEGER :: ngridx(1:3)
       double precision hsml
       integer i, j, k, d
       double precision nppg

c      Averaged number of particles per grid cell

       parameter( nppg = 5.e0 )

c      Initialize parameters: Maximum number of grid cells

       ALLOCATE (maxng(1:3),ngrid(1:3),
      &   STAT=IERR)

       maxng(1) = maxngx
       if (dim.ge.2) then
         maxng(2) = maxngy
         if (dim.eq.3) then
           maxng(3) = maxngz
         endif
       endif

       do d=1,3
         ngrid(d) = 1
       enddo
```

```
c     Range of sorting grid

      maxgridx(1) = x_maxgeom
      mingridx(1) = x_mingeom
      if (dim.ge.2) then
        maxgridx(2) = y_maxgeom
        mingridx(2) = y_mingeom
        if (dim.eq.3) then
          maxgridx(3) = z_maxgeom
          mingridx(3) = z_mingeom
        endif
      endif

      do d=1,dim
         dgeomx(d) = maxgridx(d) - mingridx(d)
      enddo


c     Number of grid cells in x-, y- and z-direction:

      if (dim.eq.1) then
        ngridx(1) = min(int(ntotal/nppg) + 1,maxng(1))
      else if (dim.eq.2) then
        ngridx(1) = min(
     &  int(sqrt(ntotal*dgeomx(1)/(dgeomx(2)*nppg))) + 1,maxng(1))
        ngridx(2) = min(
     &  int(ngridx(1)*dgeomx(2)/dgeomx(1)) + 1,maxng(2))
      else if (dim.eq.3) then
        ngridx(1) = min(int((ntotal*dgeomx(1)*dgeomx(1)/
     &       (dgeomx(2)*dgeomx(3)*nppg))**(1.e0/3.e0)) + 1,maxng(1)
        )
        ngridx(2) = min(
     &       int(ngridx(1)*dgeomx(2)/dgeomx(1)) + 1,maxng(2))
        ngridx(3) = min(
     &  int(ngridx(1)*dgeomx(3)/dgeomx(1)) + 1,maxng(3))
      endif
```

```
c       Smoothing Length measured in grid cells:

        do d=1,dim
           ghsmlx(d) = int(real(ngridx(d))*hsml/dgeomx(d)) + 1
        enddo

        do d=1,dim
          ngrid(d) = ngridx(d)
        enddo

c       Initialize grid

c$OMP PARALLEL
c$OMP do
        do i=1,ngrid(1)
          do j=1,ngrid(2)
            do k=1,ngrid(3)
              grid(i,j,k) = 0
            enddo
          enddo
        enddo
c$OMP enddo
c$OMP END PARALLEL

        end
```

### 7.3.11 input.f

```
        subroutine input(x,vx,mass,rho,p,u,itype,hsml,ntotal,xl,yl,
           km,gm,
     &                  von,nspart,totalmass,pm,Rt,Lmom,Pmom,Ibody,
     &                  invIbody,nplatelet,zl,numDens)

c
```

---

```
c     Subroutine for loading or generating initial particle
   information

c     x— coordinates of particles  [out]
c     vx— velocities of particles    [out]
c     mass— mass of particles     [out]
c     rho— dnesities of particles   [out]
c     p— pressure  of particles      [out]
c     u— internal energy of particles   [out]
c     itype— types of particles    [out]
c     hsml— smoothing lengths of particles [out]
c     ntotal— total particle number  [out]
c     km ——  Bulk Modulus   [out]
c     gm ——  Shear Modulus  [out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: itype(:)
      integer  ntotal,nprobe,nspart,pm,nplatelet
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
          (:),
     &       p(:), u(:), hsml(:), km(:), gm(:), von(:),Rt(:,:,:),
     &     Lmom(:,:),Pmom(:,:),Ibody(:,:,:),invIbody(:,:,:),
        numDens(:)
      double precision xl,yl,totalmass,zl
      integer i, d, im


      INTERFACE
          subroutine sliding_wedge(x, vx, mass, rho, p, u, itype,
     &                hsml, ntotal, xl, yl)
          REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),
     &          rho(:), p(:), u(:), hsml(:)
          INTEGER, ALLOCATABLE :: itype(:)
          INTEGER :: ntotal
          Double Precision :: xl, yl
```

```
      END SUBROUTINE

      subroutine shear_cavity(x, vx, mass, rho, p, u, itype,
&                 hsml, ntotal, km, gm, von, nspart,totalmass,
&                 pm,Rt,Lmom,Pmom,Ibody,invIbody,nplatelet,
&                 xl,yl,zl,numDens)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),
&          rho(:), p(:), u(:), hsml(:), km(:), gm(:), von(:),
&          Rt(:,:,:),Lmom(:,:),Pmom(:,:),Ibody(:,:,:),
&          invIbody(:,:,:),numDens(:)
      INTEGER, ALLOCATABLE :: itype(:)
      INTEGER :: ntotal, nspart,pm,nplatelet
      double precision :: totalmass,xl,yl,zl
      END SUBROUTINE

      subroutine water_discharge(x, vx, mass, rho, p, u,
&            itype, hsml, ntotal,km,gm,von)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),
&          rho(:), p(:), u(:), hsml(:), km(:), gm(:),von(:)
      INTEGER, ALLOCATABLE :: itype(:)
      INTEGER :: ntotal
      END SUBROUTINE

      subroutine gear_teeth(x, vx, mass, rho, p, u,
&            itype, hsml, ntotal,km,gm,von)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),
&          rho(:), p(:), u(:), hsml(:), km(:), gm(:),von(:)
      INTEGER, ALLOCATABLE :: itype(:)
      INTEGER :: ntotal
      END SUBROUTINE

 END INTERFACE


 if (shearcavity) call shear_cavity(x, vx, mass, rho, p, u,
```

```
     &                          itype, hsml, ntotal, km, gm, von,
        nspart,
     &                        totalmass,pm,Rt,Lmom,Pmom,Ibody,
        invIbody,
     &                        nplatelet,xl,yl,zl,numDens)

      end



      subroutine shear_cavity(x, vx, mass, rho, p, u,
     &                          itype, hsml, ntotal, km, gm, von,
        nspart,
     &                totalmass,pm,Rt,Lmom,Pmom,Ibody,invIbody,
        nplatelet,
     &                xl,yl,zl,numDens)

c
-----------------------------------------------------------------------


c     This subroutine is used to generate initial data for the
c     3d shear driven cavity probem or any cube domain
c     x--- coordinates of particles   [out]
c     vx--- velocities of particles     [out]
c     mass--- mass of particles    [out]
c     rho--- dnesities of particles   [out]
c     p--- pressure  of particles     [out]
c     u--- internal energy of particles   [out]
c     itype--- types of particles  [out]
c          =2   water
c     h--- smoothing lengths of particles [out]
c     ntotal--- total particle number    [out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: itype(:)
```

```
      integer  ntotal, nspart
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
     &    (:),
     &     p(:), u(:), hsml(:), km(:), gm(:),von(:),Rt(:,:,:),Lmom
     &    (:,:),
     &     Pmom(:,:),Ibody(:,:,:),invIbody(:,:,:),Rx(:,:),Ry(:,:),
     &     Rz(:,:),Rf(:,:),numDens(:)
      integer i, j, d, m, n, o, mp, np, op, k, l,pm,nplatelet
      double precision xl, yl, zl, dx, dy, dz, dist,xcent,ycent,
     &    zcent,
     &                 totalmass,pl,xangleMin,xangleMax,yangleMin
     &    ,
     &                 yangleMax,zangleMin,zangleMax,alpha,beta,
     &    gamma,
     &                 xcents(1:3,1)

      Allocate (Rx(1:3,1:3),Ry(1:3,1:3),Rz(1:3,1:3),Rf(1:3,1:3))



c     Giving mass and smoothing length as well as other data.

      m = 164
      o = 164
      n = 13

      mp = m-1
      op = o-1
      np = n-1

      ntotal = mp * np * op

      xl = 2.05268e-6
      yl = 150e-9
      zl = 2.05268e-6

      dx = xl/mp
```

```
      dy = yl/np
      dz = zl/op


      do j = 1, np
       do l = 1, op
             do i = 1, mp
             k = i +(j-1)*mp*op + (l-1)*mp
             x(1,nspart+k) = (i-1)*dx + dx/2. + (2*dx)
             x(2,nspart+k) = (j-1)*dy + dy/2. + (2*dy)
             x(3,nspart+k) = (l-1)*dz + dz/2. + (2*dz)
          enddo
       enddo
      enddo

      do i = 1, mp*np*op
        vx(1,nspart+i) = 0.
            vx(2,nspart+i) = 0.
            vx(3,nspart+i) = 0.
        rho (nspart+i) = 862
        mass(nspart+i) = dx*dy*dz*rho(nspart+i)
        numDens(nspart+i) = rho(nspart+i)/mass(nspart+i)
        p(nspart+i) = (10 * numDens(nspart+i) *
     &      (dx**3))/(rho(nspart+i))
        u(nspart+i)=357.1
        itype(nspart+i) = 2
        hsml(nspart+i) = dy
        km(nspart+i) = 0.
        gm(nspart+i) = 0.
        von(nspart+i) = 0.
      enddo

c *** 'Freeze' those SPH particles that are within the NP
        totalmass = 0
        pm = 0
c   Add central particle to NP
       if(nano.eq.3) then
```

```
      pl = 5e-9
      ntotal = ntotal + 1
      pm = pm + 1
      x(1,nspart+ntotal) = xl/2 + 2*dy
      x(2,nspart+ntotal) = yl/2 + 2*dy
      x(3,nspart+ntotal) = zl/2 + 2*dy
      vx(1,nspart+ntotal) = 0.
      vx(2,nspart+ntotal) = 0.
      vx(3,nspart+ntotal) = 0.
      rho(nspart+ntotal) = 5010
      mass(nspart+ntotal) = dx*dy*dz*rho(nspart+ntotal)
      totalmass = totalmass + mass(nspart+ntotal)
      p(nspart+ntotal) = 0.
      itype(nspart+ntotal) = 12
      hsml(nspart+ntotal) = dy

      xcent = x(1,nspart+ntotal)
      ycent = x(2,nspart+ntotal)
      zcent = x(3,nspart+ntotal)

c     Freeze particles around a certain radius around center NP

        do i = 1, ntotal - 1
          dist = sqrt((x(1,i+nspart)-xcent)**2+
     &       (x(2,i+nspart)-ycent)**2+(x(3,i+nspart)-zcent)**2)
          if(dist.le.pl) then
          pm = pm + 1
            itype(i+nspart) = 11
           rho(i+nspart) = 5010
c            rho(i+nspart) = 862
            mass(i+nspart) = dx*dy*dz*rho(i+nspart)
            totalmass = totalmass + mass(i+nspart)
          endif
        enddo
c ***  Calculate Information for rigid body rotation
       Rt(1,1,1) = 1
       Rt(1,2,1) = 0
```

```
          Rt(1,3,1) = 0
          Rt(2,1,1) = 0
          Rt(2,2,1) = 1
          Rt(2,3,1) = 0
          Rt(3,1,1) = 0
          Rt(3,2,1) = 0
          Rt(3,3,1) = 1

          Lmom(:,1) = 0
          Pmom(:,1) = 0

          Ibody(1,1,1) = (2.0/5.0)*totalmass*pl**2
          Ibody(1,2,1) = 0
          Ibody(1,3,1) = 0
          Ibody(2,1,1) = 0
          Ibody(2,2,1) = (2.0/5.0)*totalmass*pl**2
          Ibody(2,3,1) = 0
          Ibody(3,1,1) = 0
          Ibody(3,2,1) = 0
          Ibody(3,3,1) = (2.0/5.0)*totalmass*pl**2
          invIbody(1,1,1) = 1/Ibody(1,1,1)
          invIbody(1,2,1) = 0
          invIbody(1,3,1) = 0
          invIbody(2,1,1) = 0
          invIbody(2,2,1) = 1/Ibody(2,2,1)
          invIbody(2,3,1) = 0
          invIbody(3,1,1) = 0
          invIbody(3,2,1) = 0
          invIbody(3,3,1) = 1/Ibody(3,3,1)

          nplatelet = 1

        elseif(nano.eq.4) then
c   *** Convert SPH Particles to Nanosheet
          pl = 3.18e-7
          do i = 1, ntotal
             if(x(1,i).ge.((2*dx)+(xl/2)-(pl/2)).and.x(1,i).le.
```

```
     &                          ((2*dx)+(xl/2)+(pl/2))) then
                if(x(2,i).gt.((2*dy)+(yl/2)-(dy/2)).and.x(2,i).lt.
     &                          ((2*dy)+(yl/2)+(dy/2))) then
                 if(x(3,i).ge.((2*dz)+(xl/2)-(pl/2)).and.x(3,i).le.
     &                          ((2*dz)+(xl/2)+(pl/2))) then
!!     Temporarily add fluid particles to the ones made into a NS
!                  ntotal = ntotal + 1
!                  x(1,ntotal) = x(1,i)
!                  x(2,ntotal) = x(2,i)
!                  x(3,ntotal) = x(3,i)
!                  vx(1,ntotal) = 0.
!                  vx(2,ntotal) = 0.
!                  vx(3,ntotal) = 0.
!                  p(ntotal) = 0.
!                  itype(ntotal) = 2
!                  rho(ntotal) = 862
!                  mass(ntotal) = dx*dy*dz*rho(i)
!                  hsml(ntotal) = dy
                  vx(1,i) = 0
                  vx(2,i) = 0
                  vx(3,i) = 0
                   pm = pm + 1
                   itype(i) = 11
                   rho(i) = 5010
                   mass(i) = dx*dy*dz*rho(i)
                   numDens(i) = rho(i)/mass(i)
                   p(nspart+i) = (10 * numDens(nspart+i) *
     &                (dx**3))/(rho(nspart+i))
                   totalmass = totalmass + mass(i)
                    vx(1,i) = 0.
                    vx(2,i) = 0.
                    vx(3,i) = 0.
                    if(x(1,i).gt.((2*dx)+(xl/2)-(dx/2)).and.x(1,i).
     &                    le.((2*dx)+(xl/2)+(dx/2))) then
                      if(x(3,i).gt.((2*dz)+(xl/2)-(dz/2)).and.
     &                      x(3,i).le.((2*dz)+(xl/2)+(dz/2))) then
                         itype(i) = 12
```

```
                        xcents(1,1) = x(1,i)
                        xcents(2,1) = x(2,i)
                        xcents(3,1) = x(3,i)
                      endif
                    endif
                  endif
                endif
              endif
            enddo

            nplatelet = 1

c *** Setup limits for random angles
            xangleMin = 0.0
            xangleMax = .10124

            yangleMin = 0.0
            yangleMax = pi/2

            zangleMin = 0.0
            zangleMax = .10124

c         ** Calculate the rotation of each platelet

              alpha = pi/18
              beta = 0
              gamma = 0

c *** Calculate Information for rigid body rotation
              Rx(1,1) = 1
              Rx(1,2) = 0
              Rx(1,3) = 0
              Rx(2,1) = 0
              Rx(2,2) = cos(gamma)
              Rx(2,3) = -sin(gamma)
              Rx(3,1) = 0
              Rx(3,2) = sin(gamma)
```

```
Rx(3,3) = cos(gamma)
Ry(1,1) = cos(beta)
Ry(1,2) = 0
Ry(1,3) = sin(beta)
Ry(2,1) = 0
Ry(2,2) = 1
Ry(2,3) = 0
Ry(3,1) = -sin(beta)
Ry(3,2) = 0
Ry(3,3) = cos(beta)
Rz(1,1) = cos(alpha)
Rz(1,2) = -sin(alpha)
Rz(1,3) = 0
Rz(2,1) = sin(alpha)
Rz(2,2) = cos(alpha)
Rz(2,3) = 0
Rz(3,1) = 0
Rz(3,2) = 0
Rz(3,3) = 1
Rf = matmul( Rx, Ry)
Rt(:,:,1) = matmul(Rf,Rz)

Lmom(:,1) = 0
Pmom(:,1) = 0

!          write(*,*)'second moment of inertia'

Ibody(1,1,1) = ((totalmass/nplatelet)/12)*((dy**2)+pl
   **2)
Ibody(1,2,1) = 0
Ibody(1,3,1) = 0
Ibody(2,1,1) = 0
Ibody(2,2,1) = ((totalmass/nplatelet)/12)*(pl**2+pl
   **2)
Ibody(2,3,1) = 0
Ibody(3,1,1) = 0
Ibody(3,2,1) = 0
```

```
            Ibody(3,3,1) = ((totalmass/nplatelet)/12)*(pl**2+(dy
               **2))
            invIbody(1,1,1) = 1/Ibody(1,1,1)
            invIbody(1,2,1) = 0
            invIbody(1,3,1) = 0
            invIbody(2,1,1) = 0
            invIbody(2,2,1) = 1/Ibody(2,2,1)
            invIbody(2,3,1) = 0
            invIbody(3,1,1) = 0
            invIbody(3,2,1) = 0
            invIbody(3,3,1) = 1/Ibody(3,3,1)

c *** Rotate platelet to desired angles
            do i = 1, ntotal
             if(itype(i).eq.11.or.itype(i).eq.12) then
                x(:,i) = matmul(Rt(:,:,1),(x(:,i)-xcents(:,1)))+
     &                   xcents(:,1)
             endif
            enddo


       endif

      end
```

### 7.3.12 internal_force.f

```
      subroutine int_force(itimestep,dt,ntotal,nspart,hsml,mass,vx
         ,niac,
     &        rho,eta,pair_i,pair_j,dwdx,u,itype,x,t,c,p,dvxdt,
        tdsdt,
     &        dedt,txx,txy,tyy,km,gm,von,tyx,tdotxx,tdotxy,tdotyy,
        rxx,
     &        rxy,ryy,tauxx,tauxy,tauyy,rhonot,totaleta,gammadot,
     &        numDens)

c
```

```
c   Subroutine to calculate the internal forces on the right hand
    side
c   of the Navier-Stokes equations, i.e. the pressure gradient and
    the
c   gradient of the viscous stress tensor, used by the time
    integration.
c   Moreover the entropy production due to viscous dissipation,
    tds/dt,
c   and the change of internal energy per mass, de/dt, are
    calculated.

c     itimestep: Current timestep number  [in]
c     dt      :   Time step              [in]
c     ntotal  : Number of particles  [in]
c     hsml    : Smoothing Length  [in]
c     mass    : Particle masses   [in]
c     vx      : Velocities of all particles   [in]
c     niac    : Number of interaction pairs  [in]
c     rho     : Density      [in]
c     eta     : Dynamic viscosity   [in]
c     pair_i  : List of first partner of interaction pair [in]
c     pair_j  : List of second partner of interaction pair [in]
c     dwdx    : Derivative of kernel with respect to x, y and z [in
    ]
c     itype   : Type of particle (material types) [in]
c     u       : Particle internal energy  [in]
c     x       : Particle coordinates  [in]
c     itype   : Particle type       [in]
c     t       : Particle temperature   [in/out]
c     c       : Particle sound speed   [out]
c     p       : Particle pressure   [out]
c     dvxdt   : Acceleration with respect to x, y and z  [out]
c     tdsdt   : Production of viscous entropy [out]
c     dedt    : Change of specific internal energy [out]

      implicit none
```

```
 include 'param.inc'

 INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:), itype(:)
 integer itimestep, ntotal,niac, IERR,nspart
 REAL (KIND=8), ALLOCATABLE :: hsml(:), mass(:), vx(:,:), rho
     (:),
&        eta(:), dwdx(:,:), u(:), x(:,:), t(:), c(:), p(:),
&        dvxdt(:,:), tdsdt(:), dedt(:), txx(:), tyy(:),
&        tzz(:), txy(:), txz(:), tyz(:), tdotxx(:), tdotyy(:),
&        tdotxy(:),tauxx(:),tauyy(:),tauxy(:),rxx(:),rxy(:),
    ryy(:),
&        km(:),gm(:),von(:),ryx(:),tdotyx(:),tauyx(:),tyx(:),
&        rhonot(:),fpq(:,:),dpdxtot(:),atot(:),btot(:),batot
    (:),
&        bbtot(:),bctot(:),fpqnx(:,:),sep(:,:),numDens(:)
 REAL (KIND=8) :: dvx(1:3),dx(1:3),fpqt,fijc,dpdxtott,atott,
     btott,
&        batott, bbtott, bctott,fpqtnx,sept
 double precision dt,r,f,pt,totaleta,vxf,volume,xl,yl,zl,
     gammadot
 integer i, j, k, d
 double precision hxx, hyy, hzz, hxy, hxz, hyz, h, he, rhoij,
&         hrxx,hrxy,hryx,hryy,hyx,a,ba,bb,bc,dotdvx,dxs,xcent,
&         ycent,zcent,dist,distj,masstot

 ALLOCATE (txx(1:maxn),tyy(1:maxn),tzz(1:maxn),txy(1:maxn),
&     txz(1:maxn),tyz(1:maxn),tdsdt(1:maxn),
&     dedt(1:maxn),tdotxx(1:maxn),tdotyy(1:maxn),tdotxy(1:maxn
    ),
&     tauxx(1:maxn),tauyy(1:maxn),tauxy(1:maxn),rxx(1:maxn),
&     rxy(1:maxn),ryy(1:maxn),km(1:maxn),gm(1:maxn),
&     ryx(1:maxn),tdotyx(1:maxn),tauyx(1:maxn),tyx(1:maxn),
&     fpq(1:3,1:maxn),dpdxtot(1:maxn),atot(1:maxn),btot(1:maxn
    ),
&     batot(1:maxn),bbtot(1:maxn),bctot(1:maxn),fpqnx(1:3,1:
    maxn),
&     sep(1:3,1:maxn),STAT=IERR)
```

```
      xl = 0.03
      zl = 0.03
      yl = 0.03
c     Initialization of shear tensor, velocity divergence,
c     viscous energy, internal energy, acceleration


c$OMP PARALLEL
c$OMP do
      do i=1,ntotal
        txx(i) = 0.e0
        tyy(i) = 0.e0
        tzz(i) = 0.e0
        txy(i) = 0.e0
        rxx(i) = 0.e0
        ryy(i) = 0.e0
        rxy(i) = 0.e0
        txz(i) = 0.e0
        tyz(i) = 0.e0
        txz(i) = 0.e0
c         vcc(i) = 0.e0
        tdotxx(i) = 0
        tdotyy(i) = 0
        tdotxy(i) = 0
c         tdotyx(i) = 0
        tdsdt(i) = 0.e0
        dedt(i) = 0.e0
        von(i) = 0.
        dpdxtot(i) = 0.e0
        atot(i) = 0.e0
        btot(i) = 0.e0
        batot(i) = 0.e0
        bbtot(i) = 0.e0
        bctot(i) = 0.e0
        do d=1,dim
```

```
            dvxdt(d,i) = 0.e0
            fpq(d,i) = 0.e0
            fpqnx(d,i) = 0.e0
            sep(d,i) = 0.e0
          enddo
        enddo
c$OMP enddo
c$OMP end parallel

c  Initialize initial density for particles
        if(itimestep.eq.1) then
c$OMP PARALLEL
c$OMP do
          do i = nspart+1, ntotal+nspart
            if(number_density) then
               rhonot(i) = numDens(i)
            else
                rhonot(i) = rho(i)
            endif
          enddo
c$OMP enddo
c$OMP END PARALLEL
         endif

c *** Initialize stress parameters


c      Calculate SPH sum for shear tensor Tab = va,b + vb,a (
   incomrpessible fluid)
        if (visc) then
         if(pa_sph.eq.1) then

c$OMP PARALLEL
c$OMP do PRIVATE(k,i,j,d,dvx,hxx,hyy,hzz,hxy,hxz,hyz)
          do k=1,niac
            i = pair_i(k)
            j = pair_j(k)
```

```
      if(itype(i).eq.2.and.itype(j).eq.2) then
        do d=1,dim
          dvx(d) = vx(d,j) - vx(d,i)
        enddo

        if (dim.eq.1) then
         if(abs(itype(i)).eq.2.and.abs(itype(j)).eq.2) then
          hxx = 2.e0*dvx(1)*dwdx(1,k)
         elseif(mod(3,itype(i)).eq.0.and.mod(3,itype(j)).eq.0)
             then
          hxx = dvx(1)*dwdx(1,k)
         endif
        else if (dim.eq.2) then
         if(abs(itype(i)).eq.2.and.abs(itype(j)).eq.2) then
          hxx = 2.e0*dvx(1)*dwdx(1,k)
          hxy = dvx(1)*dwdx(2,k) + dvx(2)*dwdx(1,k)
          hyy = 2.e0*dvx(2)*dwdx(2,k)
        endif
        else if (dim.eq.3) then
          hxx = 2.e0*dvx(1)*dwdx(1,k) - dvx(2)*dwdx(2,k)
     &                                - dvx(3)*dwdx(3,k)
          hxy = dvx(1)*dwdx(2,k) + dvx(2)*dwdx(1,k)
          hxz = dvx(1)*dwdx(3,k) + dvx(3)*dwdx(1,k)
          hyy = 2.e0*dvx(2)*dwdx(2,k) - dvx(1)*dwdx(1,k)
     &                                - dvx(3)*dwdx(3,k)
          hyz = dvx(2)*dwdx(3,k) + dvx(3)*dwdx(2,k)
          hzz = 2.e0*dvx(3)*dwdx(3,k) - dvx(1)*dwdx(1,k)
     &                                - dvx(2)*dwdx(2,k)
        endif
        if(abs(itype(i)).eq.2.and.abs(itype(j)).eq.2) then
        hxx = 2.e0/3.e0*hxx
        hyy = 2.e0/3.e0*hyy
        hzz = 2.e0/3.e0*hzz
        endif

        if (dim.eq.1) then
```

```
            else if (dim.eq.2) then
c---- Have to be careful here when running in parallel----
c---- Flush stress variable across threads to makes sure
c---- each thread sees the same value. Then write atomically
c---- to stress variable to avoid simultaneous memory calls
c$OMP FLUSH (txx)
c$OMP ATOMIC
                txx(i) = txx(i) + mass(j)*hxx/rho(j)
c$OMP FLUSH (txx)
c$OMP ATOMIC
                txx(j) = txx(j) + mass(i)*hxx/rho(i)
c$OMP FLUSH (txy)
c$OMP ATOMIC
                txy(i) = txy(i) + mass(j)*hxy/rho(j)
c$OMP FLUSH (txy)
c$OMP ATOMIC
                txy(j) = txy(j) + mass(i)*hxy/rho(i)
c$OMP FLUSH (tyy)
c$OMP ATOMIC
                tyy(i) = tyy(i) + mass(j)*hyy/rho(j)
c$OMP FLUSH (tyy)
c$OMP ATOMIC
                tyy(j) = tyy(j) + mass(i)*hyy/rho(i)

cc$OMP FLUSH (rxx)

c                 rxx(i) = 0
cc$OMP FLUSH (rxx)

c                 rxx(j) = 0
cc$OMP FLUSH (rxy)
cc$OMP ATOMIC
c                 rxy(i) = rxy(i) + mass(j)*hrxy/rho(j)
cc$OMP FLUSH (rxy)

c                 rxy(j) = rxy(j) + mass(i)*hrxy/rho(i)
cc$OMP FLUSH (ryy)
```

```
c                 ryy(i) = 0
cc$OMP FLUSH (ryy)

c                 ryy(j) = 0

              else if (dim.eq.3) then
c$OMP FLUSH (txx)
c$OMP ATOMIC
                  txx(i) = txx(i) + mass(j)*hxx/rho(j)
c$OMP FLUSH (txx)
c$OMP ATOMIC
                  txx(j) = txx(j) + mass(i)*hxx/rho(i)
c$OMP FLUSH (txy)
c$OMP ATOMIC
                  txy(i) = txy(i) + mass(j)*hxy/rho(j)
c$OMP FLUSH (txy)
c$OMP ATOMIC
                  txy(j) = txy(j) + mass(i)*hxy/rho(i)
c$OMP FLUSH (txz)
c$OMP ATOMIC
                  txz(i) = txz(i) + mass(j)*hxz/rho(j)
c$OMP FLUSH (txz)
c$OMP ATOMIC
                  txz(j) = txz(j) + mass(i)*hxz/rho(i)
c$OMP FLUSH (tyy)
c$OMP ATOMIC
                  tyy(i) = tyy(i) + mass(j)*hyy/rho(j)
c$OMP FLUSH (tyy)
c$OMP ATOMIC
                  tyy(j) = tyy(j) + mass(i)*hyy/rho(i)
c$OMP FLUSH (tyz)
c$OMP ATOMIC
                  tyz(i) = tyz(i) + mass(j)*hyz/rho(j)
c$OMP FLUSH (tyz)
c$OMP ATOMIC
                  tyz(j) = tyz(j) + mass(i)*hyz/rho(i)
```

```
c$OMP FLUSH (tzz)
c$OMP ATOMIC
                tzz(i) = tzz(i) + mass(j)*hzz/rho(j)
c$OMP FLUSH (tzz)
c$OMP ATOMIC
                tzz(j) = tzz(j) + mass(i)*hzz/rho(i)
              endif

          endif
          enddo
c$OMP enddo
c$OMP END PARALLEL
        endif
        endif
c       Pressure from equation of state
c$OMP PARALLEL
c$OMP do
        do i=nspart+1,ntotal+nspart
          if(itype(i).eq.2.or.itype(i).eq.12.or.
     &       itype(i).eq.11) then
           call p_art_water(rho(i), p(i),rhonot(i),numDens(i),mass(i
              ))
              else if (abs(itype(i)).eq.3.or.abs(itype(i)).eq.1)
                 then
                call p_solid(rho(i), p(i), km(i))
              endif
        enddo
c$OMP enddo
c$OMP END PARALLEL


c       Calculate SPH sum for pressure force -p,a/rho
c       and viscous force (eta Tab),b/rho
c       and the internal energy change de/dt due to -p/rho vc,c


c$OMP PARALLEL
```

```
c$OMP do PRIVATE(k,i,j,d,he,rhoij,h,dx,dvx,dotdvx,dxs,ba,bb,bc,a,r
     ,f)
        do k=1,niac
          i = pair_i(k)
          j = pair_j(k)

c       For SPH algorithm 1
         if(pa_sph.eq.1) then
          rhoij = 1.e0/(rho(i)*rho(j))
            do d=1,3

c       Pressure part
                h = -(p(i) + p(j))*dwdx(d,k)
c       Viscous force
                 if (d.eq.1) then
c ---- Calculate the acceleration of the particles and determine if
     the
c ---- particle is solid or fluid for the correct equation
c       x-coordinate of acceleration
                  h = h + (eta(i)*txx(i) + eta(j)*txx(j))*dwdx(1,k)
                  if (dim.ge.2) then
                    h = h + (eta(i)*txy(i) + eta(j)*txy(j))*dwdx(2,k)
                    if (dim.eq.3) then
                      h = h + (eta(i)*txz(i) + eta(j)*txz(j))*dwdx(3,
                         k)
                    endif
                  endif
                elseif (d.eq.2) then

c       y-coordinate of acceleration
                  h = h + (eta(i)*txy(i) + eta(j)*txy(j))*dwdx(1,k)
     &                  + (eta(i)*tyy(i) + eta(j)*tyy(j))*dwdx(2,k)
                  if (dim.eq.3) then
                    h = h + (eta(i)*tyz(i) + eta(j)*tyz(j))*dwdx(3,k)
                  endif
                elseif (d.eq.3) then
```

```
c      z-coordinate of acceleration

                h = h + (eta(i)*txz(i) + eta(j)*txz(j))*dwdx(1,k)
     &                + (eta(i)*tyz(i) + eta(j)*tyz(j))*dwdx(2,k)
     &                + (eta(i)*tzz(i) + eta(j)*tzz(j))*dwdx(3,k)
              endif

            h = h*rhoij

            dvxdt(d,i) = dvxdt(d,i) + mass(j)*h
            dvxdt(d,j) = dvxdt(d,j) - mass(i)*h
           enddo
         elseif(pa_sph.eq.3) then
          if(itype(i).ne.4.and.itype(j).ne.4) then
          do d = 1,3
            dx(d) = x(d,i) - x(d,j)
            dvx(d) = vx(d,i) - vx(d,j)
          enddo
           r = sqrt(dx(1)*dx(1)+dx(2)*dx(2)+dx(3)*dx(3))
           dotdvx = dx(1)*dvx(1) + dx(2)*dvx(2) + dx(3)*dvx(3)
           dxs = dx(1)*dx(1)+dx(2)*dx(2)+dx(3)*dx(3)
           f = (315/(4*pi*(2*hsml(i))**5))*(1-(r/(2*hsml(i))))**2
c           if(i.eq.1) then
c              write(*,*)f
c           endif
          do d = 1,3
c        Pressure Part
          if(number_density) then
             h = -((p(i)/(numDens(i)*mass(i))**(2.0))+
     &            (p(j)/(numDens(j)*mass(j))**(2.0)))*dwdx(d,k)
          else
             h = -((p(i)/rho(i)**(2.0))+(p(j)/rho(j)**(2.0)))*dwdx(
                d,k)
          endif
!          h = ((p(i)/rho(i)**(2.0))+(p(j)/rho(j)**(2.0)))*f*dx(d)
```

```
c       Viscous Part
        if(dx(d).ne.0) then
         if(number_density) then
           a = (((eta(i)+eta(j))*dx(d)*dvx(d))/dxs)*
     &          (dwdx(d,k)/(numDens(j)*mass(j)))
           bc = (((5.0)*((dx(d)*dotdvx)/dxs))-dvx(d))*
     &          ((dwdx(d,k)/dx(d))/(numDens(j)*mass(j)))
         else
          a = (((eta(i)+eta(j))*dx(d)*dvx(d))/dxs)*
     &          (dwdx(d,k)/rho(j))
          bc = (((5.0)*((dx(d)*dotdvx)/dxs))-dvx(d))*
     &          ((dwdx(d,k)/dx(d))/rho(j))
         endif
          ba = dwdx(d,k)*(5.0/3.0)*(eta(j)-eta(i))
          bb = (dwdx(d,k))*dvx(d)
        else
          a = 0
          ba = 0
          bb = 0
          bc = 0
        endif

          if(number_density) then
           dvxdt(d,i) = dvxdt(d,i) + (((1.e0)/(numDens(i)
     &          *mass(i)))*((mass(j)*(numDens(i)*mass(i))*h)+
     &          ((5.0)*a*mass(j))-(mass(j)*((5.0/3.0)*eta(j)*bc))
     &          +(mass(j)**2*(1/(numDens(i)*mass(i)))**2*ba*bb)))

            dvxdt(d,j) = dvxdt(d,j) - (((1.e0)/(numDens(j)
     &          *mass(j)))*((mass(i)*(numDens(j)*mass(j))*h)+
     &          ((5.0)*a*mass(i))-(mass(i)*((5.0/3.0)*eta(i)*bc))
     &          +(mass(i)**2*(1/(numDens(j)*mass(j)))**2*ba*bb)))

          else

           dvxdt(d,i) = dvxdt(d,i) + (((1.e0)/rho(i))*
     &          ((mass(j)*rho(i)*h)+((5.0)*a*mass(j))-(mass(j)*
```

```
     &             ((5.0/3.0)*eta(j)*bc))+(mass(j)**2*(1/rho(i))**2
     &             *ba*bb)))

              dvxdt(d,j) = dvxdt(d,j) - (((1.e0)/rho(j))*
     &             ((mass(i)*rho(j)*h)+((5.0)*a*mass(i))-(mass(i)*
     &             ((5.0/3.0)*eta(i)*bc))+(mass(i)**2*(1/rho(j))**2
     &             *ba*bb)))
              endif

c ***   Calculate local Stress Tensor for determining total
   viscosity

c --- Don't include fluid particles within the nanoparticle

              fpq(d,i) = fpq(d,i) + mass(j)*(((1.e0)/rho(i))*
     &             ((mass(j)*rho(i)*h)+((5.0)*a*mass(j))-(mass(j)*
     &             ((5.0/3.0)*eta(j)*bc))+(mass(j)**2*(1/rho(i))**2
     &             *ba*bb)))*(x(2,i)-x(2,j))


!             dist = sqrt((x(1,i)-xcent)**2+(x(2,i)-ycent)**2+
!     &                    (x(3,i)-zcent)**2)
!             if(dist.gt.26e-9) then
              fpq(d,j) = fpq(d,j) - mass(i)*(((1.e0)/rho(j))*
     &             ((mass(i)*rho(j)*h)+((5.0)*a*mass(i))-(mass(i)*
     &             ((5.0/3.0)*eta(i)*bc))+(mass(i)**2*(1/rho(j))**2
     &             *ba*bb)))*(x(2,j)-x(2,i))

            enddo

          endif
        endif

       enddo
c$OMP enddo
c$OMP END PARALLEL
```

```
        fpqt = 0.0
        pt = 0.0
        fijc = 0.0
        masstot = 0.0
c ***   Calculate stress tensor due to propagation of momentum and
c       inter-particle forces

       if(mod(itimestep,save_step).eq.0) then
        do i = 1, ntotal
         if(itype(i).eq.2.or.itype(i).eq.11.or.itype(i).eq.12)then
            fpqt = fpqt + fpq(1,i)
            fpqtnx = fpqtnx + fpqnx(1,i)
            sept = sept + sep(2,i)
            vxf = .0056*(1-((x(2,i)-(2*hsml(i)))/(yl)))
            pt = pt + mass(i)**2*(vx(1,i)-vxf)*vx(2,i)

            dpdxtott = dpdxtott + dpdxtot(i)
            atott = atott + atot(i)
            btott = btott + btot(i)
            batott = batott + batot(i)
            bbtott = bbtott + bbtot(i)
            bctott = bctott + bctot(i)
            masstot = masstot + mass(i)
          endif
        enddo

        fpqt = fpqt / (2 * ((xl)*(yl)*(zl)))
        pt = pt / ((xl)*(yl)*(zl)*masstot)

        volume = 1 / (2 * ((xl)*(yl)*(zl)))

        if(spart) then
         totaleta = fpqt + pt
        else
c ***   Calculate total viscosity
```

```
c ---   Add correction factor to make viscosity match isoviscous
   simulation
        totaleta = (fpqt + pt)/gammadot

           open(75,file='viscosity.txt')
           write(75, *) itimestep, totaleta
           write(*,*)'total viscosity:', totaleta
         endif
        endif

      end
```

### 7.3.13 kernel.f

```
      subroutine kernel(r,dx,hsml,w,dwdx)

c
   ------------------------------------------------------------------


c   Subroutine to calculate the smoothing kernel wij and its
c   derivatives dwdxij.
c     if skf = 1, cubic spline kernel by W4 - Spline (Monaghan
   1985)
c            = 2, Gauss kernel   (Gingold and Monaghan 1981)
c            = 3, Quintic kernel (Morris 1997)

c     r    : Distance between particles i and j    [in]
c     dx   : x-, y- and z-distance between i and j  [in]
c     hsml : Smoothing length  [in]
c     w    : Kernel for all interaction pairs [out]
c     dwdx : Derivative of kernel with respect to x, y and z  [out
   ]

      implicit none
      include 'param.inc'

      REAL (KIND=8) :: dwdx(1:3),dx(1:3)
      double precision r, hsml, w
```

```
integer i, j, d
double precision q, dw, factor

q = r/hsml
w = 0.e0
do d=1,dim
  dwdx(d) = 0.e0
enddo

if (skf.eq.1) then

  if (dim.eq.1) then
    factor = 1.e0/hsml
  elseif (dim.eq.2) then
    factor = 15.e0/(7.e0*pi*hsml*hsml)
  elseif (dim.eq.3) then
    factor = 3.e0/(2.e0*pi*hsml*hsml*hsml)
  else
   print *,' >>> Error <<< : Wrong dimension: Dim =',dim
   stop
  endif
  if (q.ge.0.and.q.le.1.e0) then
    w = factor * (2./3. - q*q + q**3 / 2.)
    do d = 1, dim
      dwdx(d) = factor * (-2.+3./2.*q)/hsml**2 * dx(d)
    enddo
  else if (q.gt.1.e0.and.q.le.2) then
    w = factor * 1.e0/6.e0 * (2.-q)**3
    do d = 1, dim
      dwdx(d) =-factor * 1.e0/6.e0 * 3.*(2.-q)**2/hsml * (dx
         (d)/r)
    enddo
  else
    w=0.
    do d= 1, dim
      dwdx(d) = 0.
    enddo
```

```
    endif

  else if (skf.eq.2) then

    factor = 1.e0 / (hsml**dim * pi**(dim/2.))
    if(q.ge.0.and.q.le.3) then
      w = factor * exp(-q*q)
      do d = 1, dim
        dwdx(d) = w * ( -2.* dx(d)/hsml/hsml)
      enddo
    else
      w = 0.
      do d = 1, dim
        dwdx(d) = 0.
      enddo
    endif

  else if (skf.eq.3) then

    if (dim.eq.1) then
      factor = 1.e0 / (120.e0*hsml)
    elseif (dim.eq.2) then
      factor = 7.e0 / (478.e0*pi*hsml*hsml)
    elseif (dim.eq.3) then
      factor = 1.e0 / (120.e0*pi*hsml*hsml*hsml)
    else
     print *,' >>> Error <<< : Wrong dimension: Dim =',dim
     stop
    endif
    if(q.ge.0.and.q.le.1) then
      w = factor * ( (3-q)**5 - 6*(2-q)**5 + 15*(1-q)**5 )
      do d= 1, dim
        dwdx(d) = factor * ( (-120 + 120*q - 50*q**2)
&                          / hsml**2 * dx(d) )
      enddo
    else if(q.gt.1.and.q.le.2) then
      w = factor * ( (3-q)**5 - 6*(2-q)**5 )
```

```
        do d= 1, dim
          dwdx(d) = factor * (-5*(3-q)**4 + 30*(2-q)**4)
     &                     / hsml * (dx(d)/r)
        enddo
      else if(q.gt.2.and.q.le.3) then
        w = factor * (3-q)**5
        do d= 1, dim
          dwdx(d) = factor * (-5*(3-q)**4) / hsml * (dx(d)/r)
        enddo
      else
        w = 0.
        do d = 1, dim
          dwdx(d) = 0.
        enddo
      endif

      endif

      end
```

### 7.3.14 link_list.f

```
      subroutine link_list(itimestep, ntotal,hsml,x,niac,pair_i,
     &           pair_j,w,dwdx,countiac,itype,nspart)

c
-------------------------------------------------------------------

c   Subroutine to calculate the smoothing funciton for each
    particle and
c   the interaction parameters used by the SPH algorithm.
    Interaction
c   pairs are determined by using a sorting grid linked list

c     itimestep : Current time step  [in]
c     ntotal    : Number of particles  [in]
c     hsml      : Smoothing Length, same for all particles  [in]
c     x         : Coordinates of all particles  [in]
```

```
c     niac       : Number of interaction pairs [out]
c     pair_i     : List of first partner of interaction pair [out]
c     pair_j     : List of second partner of interaction pair [out]
c     w          : Kernel for all interaction pairs [out]
c     dwdx       : Derivative of kernel with respect to x, y and z
    [out]
c     countiac   : Number of neighboring particles  [out]
      use omp_lib
      implicit none
      include 'param.inc'


c     Parameter used for sorting grid cells in the link list
   algorithm
c     maxngx  : Maximum number of sorting grid cells in x-
   direction
c     maxngy  : Maximum number of sorting grid cells in y-
   direction
c     maxngz  : Maximum number of sorting grid cells in z-
   direction
c     Determining maximum number of sorting grid cells:
c     (For an homogeneous particle distribution:)
c     1-dim. problem: maxngx = maxn ,  maxngy = maxngz = 1
c     2-dim. problem: maxngx = maxngy ~ sqrt(maxn) ,  maxngz = 1
c     3-dim. problem: maxngx = maxngy = maxngz ~ maxn^(1/3)
      integer maxngx,maxngy,maxngz,IERR
      parameter ( maxngx  = 160        ,
     &            maxngy  = 16       ,
     &            maxngz  = 160         )

      INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:), countiac(:),
     &        grid(:,:,:), xgcell(:,:), celldata(:),
     &        ghsmlx(:),itype(:),ppair_i(:),ppair_j(:)

      INTEGER :: minxcell(1:3),maxxcell(1:3),dnxgcell(1:3),
     &         pniac(1:24),dpxgcell(1:3),ngridx(1:3),gcell(1:3)
      REAL (KIND=8):: xs(1:3),tdwdx(1:3),dx(1:3),maxgridx(1:3),
```

```
&           mingridx(1:3),dgeomx(1:3)
 REAL (KIND=8), ALLOCATABLE :: x(:,:), w(:), dwdx(:,:)
 integer itimestep, ntotal, niac, nspart

 INTERFACE
    SUBROUTINE init_grid(ntotal,hsml,grid,ngridx,ghsmlx,
       maxgridx,
+                         mingridx,dgeomx)
    INTEGER, ALLOCATABLE :: grid(:,:,:), ghsmlx(:)
    REAL (KIND = 8) :: maxgridx(1:3),mingridx(1:3),dgeomx
       (1:3)
    INTEGER :: ngridx(1:3)
    integer :: ntotal
    double precision hsml
    END SUBROUTINE init_grid

   SUBROUTINE grid_geom(i,xs,ngridx,maxgridx,mingridx,dgeomx,
      gcell)
    integer i
    REAL (KIND = 8) :: xs(1:3)
    REAL (KIND = 8) :: dgeomx(1:3),maxgridx(1:3),mingridx
       (1:3)
    INTEGER :: ngridx(1:3),gcell(1:3)
    END SUBROUTINE

    SUBROUTINE kernel(r,dx,hsml,w,tdwdx)
    double precision r, hsml, w
    REAL (KIND=8) :: tdwdx(1:3),dx(1:3)
    END SUBROUTINE
 END INTERFACE

 double precision hsml,xl
 integer i,j,d,scale_k, sumiac, maxiac, noiac, miniac, maxp,
    minp
 integer xcell,ycell,zcell,sos
 double precision hsml2,dr,r
```

```
      ALLOCATE (ghsmlx(1:3),xgcell(1:3,1:maxn),
     &    celldata(maxn),
     &    grid(maxngx,maxngy,maxngz),STAT=IERR)


      if (skf.eq.1) then
        scale_k = 2
      else if (skf.eq.2) then
        scale_k = 3
      else if (skf.eq.3) then
         scale_k = 3
      else if (skf.eq.4) then
         scale_k = 1
      endif

c$OMP PARALLEL
C$OMP do
      do i=1,ntotal+nspart
        countiac(i) = 0
      enddo
c$OMP enddo
c$OMP END PARALLEL

c     Initialize grid:

c      write(*,*) 'Initializing the grid'
      call init_grid(ntotal,hsml,grid,ngridx,ghsmlx,
     &     maxgridx,mingridx,dgeomx)
c      write(*,*) 'Finished Initializing the grid'

c     Position particles on grid and create linked list:

c      write(*,*) 'Positioning the Particles'
      do i=1,ntotal+nspart
       if(itype(i).ne.5) then
        do j=1,dim
          xs(j) = x(j,i)
```

```
          enddo
          call grid_geom(i,xs,ngridx,maxgridx,mingridx,dgeomx,gcell)
          do d=1,dim
            xgcell(d,i) = gcell(d)
          enddo
          celldata(i) = grid(gcell(1),gcell(2),gcell(3))
c         if(i.gt.celldata(i)) then
            grid(gcell(1),gcell(2),gcell(3)) = i
c           endif
         endif
        enddo
c        write(*,*) 'Finished positioning the Particles'


c       Determine interaction parameters:
c        write(*,*) 'Searching grid'
        niac = 0

        do i=1,ntotal+nspart-1

c       Determine range of grid to go through:

          do d=1,3
            minxcell(d) = 1
            maxxcell(d) = 1
          enddo
          do d=1,dim
            dnxgcell(d) = xgcell(d,i) - ghsmlx(d)
            dpxgcell(d) = xgcell(d,i) + ghsmlx(d)
            minxcell(d) = max(dnxgcell(d),1)
            maxxcell(d) = min(dpxgcell(d),ngridx(d))
          enddo

c       Search grid:

          do zcell=minxcell(3),maxxcell(3)
            do ycell=minxcell(2),maxxcell(2)
```

```
            do xcell=minxcell(1),maxxcell(1)
              j = grid(xcell,ycell,zcell)
 1            if (j.gt.i) then
                dx(1) = x(1,i) - x(1,j)
                dr    = dx(1)*dx(1)
                do d=2,dim
                  dx(d) = x(d,i) - x(d,j)
                  dr    = dr + dx(d)*dx(d)
                enddo
                if (sqrt(dr).lt.scale_k*hsml) then
                  if (niac.lt.max_interaction) then
c     Neighboring pair list, and totalinteraction number and
c     the interaction number for each particle

                    niac = niac + 1

                    pair_i(niac) = i
                    pair_j(niac) = j


                    r = sqrt(dr)
                    countiac(i) = countiac(i) + 1
                    countiac(j) = countiac(j) + 1


C---- Kernel and derivations of kernel

                    call kernel(r,dx,hsml,w(niac),tdwdx)
                    do d = 1, dim
                      dwdx(d,niac)=tdwdx(d)
                    enddo
                  else
                    print *,
     &              ' >>> Error <<< : too many interactions',niac
                    stop
                  endif
                endif
```

```
                j = celldata(j)
                goto 1
              endif
            enddo
          enddo
        enddo

c *** Remove zeros from pair list ***

        sumiac = 0
        maxiac = 0
        miniac = 1000
        noiac  = 0
        maxp = 0
        minp = 0
        do i=1,ntotal
          sumiac = sumiac + countiac(i)
          if (countiac(i).gt.maxiac) then
            maxiac = countiac(i)
            maxp = i
          endif
          if (countiac(i).lt.miniac) then
            miniac = countiac(i)
            minp = i
          endif
          if (countiac(i).eq.0)       noiac  = noiac + 1
        enddo
         if (mod(itimestep,print_step).eq.0) then

           print *,' >> Statistics: interactions per particle:'
           print *,'**** Particle:',maxp, ' maximum interactions:',
              maxiac
           print *,'**** Particle:',minp, ' minimum interactions:',
              miniac
           print *,'**** Average :',real(sumiac)/real(ntotal)
           print *,'**** Total pairs : ',niac
```

```
        print *,'**** Particles with no interactions:',noiac

      endif

      end
```

### 7.3.15 output_recovery.f

```
      subroutine output_recovery (x, vx, rho, p, mass, itype,
         ntotal,
     &                        nspart,hsml, itimestep,time,ngrab,
        dvx,
     &                        nvirt)

c
-----------------------------------------------------------------------


c       Subroutine for outputting position, velocity, density,
c       pressure, and mass for each particle. The file generated
   is a
c       .txt file that can be read by the SPH program to restart a
    simulation
c     in the middle of the simulation time.

c       x - coordinates of particles    [in]
c       vx - velocities of particles    [in]
c       mass - mass of particles        [in]
c       rho - densities of particles    [in]
c       p - pressure of particles       [in]
c       itype - types of particles      [in]
c       ntotal - total particle number [in]
c
-----------------------------------------------------------------------


      implicit none
      include 'param.inc'
```

```
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
         (:),
     &                       p(:), hsml(:), dvx(:,:)
      INTEGER, ALLOCATABLE :: itype(:)
        integer i, itimestep
        INTEGER :: ntotal, nvirt, ngrab, nspart
        character name*17
        double precision time

c     Create a file to be used as the recovery file
      name='Recovery_file.txt'
      open(25,file=name)

c     First write the time step that the following information is
   associated with
      write(25, *) itimestep

c     Write the time
      write(25, *) time

c     Write the last .VTU file number
      write(25, *) ngrab

c     Write total number of particles
      write(25, *) ntotal

c     Write total number of solid particles
      write(25, *) nspart

c     Write total number of virtual particles
      write(25, *) nvirt

c     Write particle positions
      do i = 1, ntotal+nspart+nvirt
         write(25, *) x(1,i), x(2,i), x(3,i)
      enddo
```

```
c       Write particle velocities
        do i = 1, ntotal+nspart+nvirt
           write(25, *) vx(1,i), vx(2,i), vx(3,i)
        enddo

c       Write particle densities
        do i = 1, ntotal+nspart+nvirt
           write(25, *) rho(i)
        enddo

c       Write particle mass
        do i = 1, ntotal+nspart+nvirt
           write(25, *) mass(i)
        enddo

c       Write particle pressure
        do i = 1, ntotal+nspart+nvirt
           write(25, *) p(i)
        enddo

c       Write particle itypes
        do i = 1, ntotal+nspart+nvirt
           write(25, *) itype(i)
        enddo

c       Write particle smoothing lengths
        do i = 1, ntotal+nspart+nvirt
           write(25, *) hsml(i)
        enddo

c       Write the particle accelerations
        do i = 1, ntotal+nspart+nvirt
           write(25, *) dvx(1,i), dvx(2,i), dvx(3,i)
        enddo

        close(25)
```

```
      end
```

### 7.3.16 output_vtu.f

```
      subroutine output_vtu (x, vx, rho, p, mass, itype, ntotal,
          ngrab,
     &                       imark,von,nvirt,nspart,eta,srate,
     &                       numDens)

c
-----------------------------------------------------------------------

c       Subroutine for outputting position, velocity, density,
c       pressure, and mass for each particle. The file generated
   is a
c       .vtu file written in the xml format, which can be read by
c       numerous visualizers including Paraview.

c       x - coordinates of particles      [in]
c       vx - velocities of particles      [in]
c       mass - mass of particles          [in]
c       rho - densities of particles      [in]
c       p - pressure of particles         [in]
c       itype - types of particles        [in]
c       ntotal - total particle number [in]
c
-----------------------------------------------------------------------


      implicit none
      include 'param.inc'

     REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
         (:),
     &                    p(:), von(:),eta(:),srate(:),numDens(:)
     INTEGER, ALLOCATABLE :: itype(:), imark(:)
          integer i
     double precision vc
```

```
        INTEGER :: ntotal, ngrab, nvirt, nspart
        character name*13, supp*4

c   Determine velocity in x direction of NS or NP
c$OMP PARALLEL
c$OMP do
       do i = 1, ntotal
         if(itype(i).eq.12) then
            vc = vx(1,i)
          endif
        enddo
c$OMP enddo
c$OMP end parallel

       if(nano.eq.2) then
          vc = 0.0
       endif


c       Create a file named Part_XXXX.vtu where XXXX is a
   sequential number
        write(supp, '(i4.4)') ngrab
        name='PART_'//supp// '.vtu'
        open(13,file=name)

c       Write the appropriate header for the vtu file
        write(13, '(a)')'<?xml version="1.0"?>'
        write(13, '(a)') '<VTKFile type= "UnstructuredGrid"
           version= "0.1"
     +       byte_order= "LittleEndian">'
        write(13, *) '<UnstructuredGrid>'
        write(13,'(A,i7.7)',advance='no')'<Piece NumberOfPoints
           ="',ntotal
        write(13, '(A,i7.7)', advance='no') '" NumberOfCells="',
           ntotal

        write(13, *) '">'
```

```
c       Write the data for Pressure
        write(13,*)'   <PointData Scalars="Presure" Vectors="
           Velocity">'
        write(13,*)'     <DataArray type="Float64" Name="Pressures
           "
     + Format="ascii">'
        do i = 1, ntotal
c         if(itype(i).eq.11.or.itype(i).eq.12) then
          write(13,*) p(i)
c         endif
        enddo

        write(13,*)'     </DataArray>'

c       Write the data for Density
        write(13,*)'     <DataArray type="Float64" Name="Density"
     + Format="ascii">'
        do i = 1, ntotal
c         if(itype(i).eq.11.or.itype(i).eq.12) then
          write(13, *) rho(i)
c         endif
        enddo

        write(13,*)'      </DataArray>'

c   Write the data for numDens
        write(13,*)'       <DataArray type="Float64" Name="numDens
           "
     +                  Format="ascii">'
        do i = 1, ntotal
          write(13,*) numDens(i)
        enddo

        write(13,*)'       </DataArray>'

c       Write the data for viscosity
```

```
        write(13,*)'      <DataArray type="Float64" Name="Viscosity
           "
     + Format="ascii">'
        do i = 1, ntotal
c         if(itype(i).eq.11.or.itype(i).eq.12) then
          write(13,*) eta(i)
c         endif
        enddo

        write(13,*)'      </DataArray>'

c       Write the data for Particle information
        write(13,*)'      <DataArray type="Float64" Name="
           Scalarplot"
     + Format="ascii">'
        do i = 1, ntotal
c         if(itype(i).eq.11.or.itype(i).eq.12) then
           write(13,*) itype(i)
c         endif
        enddo

        write(13,*)'      </DataArray>'


c       Write the data for Velocity
        write(13,*)'    <DataArray type="Float64" Name="Velocity"
     + NumberOfComponents="3" Format="ascii">'
        do i = 1, ntotal
            write(13,*) vx(1,i), vx(2,i), vx(3,i)
        enddo

        write(13,*)'      </DataArray>'
        write(13,*)'    </PointData>'


c       Write the appropriate header for the section giving
   particle locations
```

```
      write(13, *)'   <Points>'
      write(13, *)'     <DataArray type="Float64"
        NumberOfComponents="3"
     + Format="ascii">'

c     Write particle locations in x, y, z coordinates
      do i = 1, ntotal
c     if(itype(i).eq.11.or.itype(i).eq.12) then
        write(13,*) x(1,i), x(2,i), x(3,i)
c      endif
      enddo


      write(13, *)' </DataArray>'
      write(13, *)' </Points>'

      write(13, *)' <Cells>'
      write(13, *)' <DataArray type="Int32" Name="connectivity"
     + Format="ascii">'

c     The following is necessary to make a complete vtu file
c     but gives no real information of the problem.
        do i = 1, ntotal
           write(13, *) i-1
        enddo

      write(13, *) '</DataArray>'
      write(13, *) '<DataArray type="Int32" Name="offsets"
     + Format="ascii">'

      do i = 1, ntotal
        write(13, *) i
      enddo


      write(13, *) '</DataArray>'
```

```
      write(13, *) '<DataArray type="Int32" Name="types"
     + Format="ascii">'


      do i = 1, ntotal
        write(13, *) (itype(i)+2)*(1/4)+1
      enddo

      write(13, *)'</DataArray>'
      write(13, *) '</Cells>'


      write(13, *)' </Piece>'
      write(13, *)' </UnstructuredGrid>'
      write(13, *)' </VTKFile>'

      end
```

### 7.3.17 single_step.f

```
       subroutine single_step(itimestep, dt, ntotal, hsml, mass, x,
          vx,
     &      u, s,rho, p, t, tdsdt, dx, dvx, du, ds, drho,itype, av,
        nvirt,
     &      xl,yl,nghost,v_min,txx,txy,tyy,imark,km,gm,von,tyx,
        tdotxx,
     &      tdotxy,tdotyy,rxx,rxy,ryy,tauxx,tauxy,tauyy,nspart,ap,
     &      spartent,nentrained,solidx,imom,wrad,angvel,totalmass,
     &      nplatelet,pm,Rt,Lmom,Pmom,Ibody,invIbody,omegaMat,vc,
        rhonot,
     &      srate,eta,numDens)

c
      ------------------------------------------------------------------


c     Subroutine to determine the right hand side of a differential
c     equation in a single step for performing time integration
```

```
c   In this routine and its subroutines the SPH algorithms are
    performed.
c     itimestep: Current timestep number  [in]
c     dt       : Timestep     [in]
c     ntotal   :  Number of particles  [in]
c     hsml     :  Smoothing Length   [in]
c     mass     :  Particle masses [in]
c     x        :  Particle position  [in]
c     vx       :  Particle velocity   [in]
c     u        :  Particle internal energy   [in]
c     s        :  Particle entropy (not used here)[in]
c     rho      :  Density   [in/out]
c     p        :  Pressure  [out]
c     t        :  Temperature  [in/out]
c     tdsdt    :  Production of viscous entropy t*ds/dt [out]
c     dx       :  dx = vx = dx/dt  [out]
c     dvx      :  dvx = dvx/dt, force per unit mass  [out]
c     du       :  du  = du/dt  [out]
c     ds       :  ds  = ds/dt    [out]
c     drho     :  drho =  drho/dt   [out]
c     itype    :  Type of particle [in]
c     av       :  Monaghan average velocity [out]

      implicit none
      include 'param.inc'

      INTEGER, ALLOCATABLE :: itype(:), pair_i(:),pair_j(:),ns(:),
     &     imark(:), spartent(:)
      integer itimestep, ntotal ,nghost,nprobe,IERR, nspart,
         nentrained,
     &        nplatelet,pm
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
         (:),
     &      p(:),u(:), s(:), hsml(:), t(:), tdsdt(:), dx(:,:), dvx
         (:,:),
     &     du(:),ds(:),drho(:),av(:,:),v_min(:,:),txx(:), tyy(:),
         txy(:),
```

```
&      tyx(:),tdotxx(:),tdotyy(:),tdotxy(:),rxx(:),rxy(:),ryy
    (:),
&      tauxx(:),tauxy(:),tauyy(:),solidx(:,:),Rt(:,:,:),Lmom
    (:,:),
&      Pmom(:,:) ,Ibody(:,:,:),invIbody(:,:,:),omegaMat(:,:,:)
    ,
&      vc(:,:),rhonot(:),srate(:),numDens(:)
 double precision dt,ap,totalmass,gammadot
 integer i, d, nvirt, niac,j, count
 REAL (KIND=8), ALLOCATABLE :: w(:), dwdx(:,:), indvxdt(:,:),
&       exdvxdt(:,:), ardvxdt(:,:), avdudt(:), ahdudt(:), c(:)
    ,
&       eta(:), psitype(:), km(:), gm(:), von(:), asdvxdt(:,:)
    ,
&       partdvxdt(:,:),wrad(:),angvel(:)
 double precision xl,yl,imom,mfactor,epsilon, sigma,zl,dist,
&       fpottemp,totaleta
 INTERFACE
      SUBROUTINE virt_part(itimestep,ntotal,nvirt,hsml,mass,x,
          vx,
&          rho,u,p,itype,nspart)
      integer itimestep,ntotal,nvirt,nspart
      REAL (KIND=8), ALLOCATABLE :: hsml(:), mass(:), x(:,:),
&             vx(:,:),rho(:), u(:), p(:)
      INTEGER, ALLOCATABLE :: itype(:)
      END SUBROUTINE

      SUBROUTINE link_list(itimestep,ntotal,hsml,x,niac,pair_i
          ,
&               pair_j,w,dwdx,ns,itype,nspart)
      integer itimestep,ntotal,niac,nspart
      double precision hsml
      REAL (KIND=8), ALLOCATABLE :: x(:,:),w(:),dwdx(:,:)
      INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:), ns(:),
          itype(:)
      END SUBROUTINE
```

```
      SUBROUTINE sum_density(ntotal,hsml,mass,niac,pair_i,
&                pair_j,w,itype,rho,numDens)
      integer ntotal,niac
      REAL (KIND=8), ALLOCATABLE :: hsml(:),mass(:),w(:),
&                rho(:),numDens(:)
      INTEGER, ALLOCATABLE :: pair_i(:),pair_j(:),itype(:),
&                imark(:)
      END SUBROUTINE

      SUBROUTINE con_density(ntotal,mass,niac,pair_i,pair_j,
&                dwdx,vx,itype,drho,imark)
      integer ntotal, niac
      REAL (KIND=8), ALLOCATABLE :: mass(:),dwdx(:,:),vx(:,:),
&                drho(:)
      INTEGER, ALLOCATABLE :: pair_i(:),pair_j(:),itype(:),
     imark(:)
      END SUBROUTINE

      SUBROUTINE viscosity(ntotal,itype,x,rho,eta,niac,pair_i,
&              pair_j,dwdx,vx,mass,srate,itimestep,nspart,
    gammadot)
      integer ntotal,niac,itimestep,nspart
      INTEGER, ALLOCATABLE :: itype(:),pair_i(:),pair_j(:)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), rho(:), eta(:),
&               dwdx(:,:),vx(:,:),mass(:),srate(:)
      double precision gammadot
      END SUBROUTINE

      SUBROUTINE int_force(itimestep,dt,ntotal,nspart,hsml,
     mass,vx,
&          niac,rho,eta,pair_i,pair_j,dwdx,u,itype,x,t,c,p,
    indvxdt,
&          tdsdt,du,txx,txy,tyy,km,gm,von,tyx,tdotxx,tdotxy,
    tdotyy,
&          rxx,rxy,ryy,tauxx,tauxy,tauyy,rhonot,totaleta,
    gammadot,
&          numDens)
```

```
      integer itimestep, ntotal, niac,nspart
      double precision dt,totaleta,gammadot
      REAL (KIND=8), ALLOCATABLE :: hsml(:), mass(:), vx(:,:),
&        rho(:),eta(:), dwdx(:,:), u(:), x(:,:), t(:), c(:),p
    (:),
&        indvxdt(:,:), tdsdt(:), du(:), txx(:), tyy(:),
&        txy(:),km(:),gm(:),von(:),tyx(:),tdotxx(:),tdotyy(:),
&        tdotxy(:),rxx(:),rxy(:),ryy(:),tauxx(:),tauxy(:),
    tauyy(:),
&        rhonot(:),numDens(:)
      INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:),itype(:)
      END SUBROUTINE

      SUBROUTINE art_visc(ntotal,hsml,
&       mass,x,vx,niac,rho,pair_i,pair_j,w,dwdx,ardvxdt,itype)

      integer ntotal, niac
      REAL (KIND=8), ALLOCATABLE :: hsml(:),mass(:),x(:,:),vx
          (:,:),
&         rho(:), w(:), dwdx(:,:), ardvxdt(:,:),c(:)
      INTEGER, ALLOCATABLE :: pair_i(:), pair_j(:),itype(:)
      END SUBROUTINE

      SUBROUTINE  art_stress(ntotal,hsml,mass,x,niac,
&      rho,pair_i,pair_j,w,dwdx,asdvxdt,tauxx,tauxy,tauyy,
    itimestep,
&      itype)
      integer ntotal,niac,itimestep
      REAL (KIND=8), ALLOCATABLE :: hsml(:),mass(:),x(:,:),
          rho(:),
&       w(:),dwdx(:,:),asdvxdt(:,:),tauxx(:),tauxy(:),tauyy(:)
      INTEGER, ALLOCATABLE :: pair_i(:),pair_j(:),itype(:)
      END SUBROUTINE

      SUBROUTINE ext_force(ntotal,mass,x,niac,pair_i,pair_j,
&         itype,hsml,exdvxdt,numDens,dwdx,p,itimestep)
      integer ntotal,niac,nvirt,itimestep
```

```
      REAL (KIND=8), ALLOCATABLE :: mass(:),x(:,:), hsml(:),
&          exdvxdt(:,:),numDens(:),dwdx(:,:),p(:)
      INTEGER, ALLOCATABLE :: itype(:),pair_i(:),pair_j(:)
      END SUBROUTINE

      SUBROUTINE av_vel(ntotal,mass,niac,pair_i,pair_j,w,vx,
     rho,
&           av,itype,imark,x,hsml,numDens)
      integer ntotal,niac
      REAL (KIND=8), ALLOCATABLE :: mass(:), w(:), vx(:,:),
     rho(:),
&          av(:,:),x(:,:),hsml(:),numDens(:)
      INTEGER, ALLOCATABLE :: itype(:), pair_i(:),pair_j(:),
     imark(:)
      END SUBROUTINE

      SUBROUTINE solid_part(x,vx,rho,p,mass,itype,ntotal,
     nprobe,
&          nspart,itimestep,hsml,partdvxdt,ap,nvirt,spartent,
&          nentrained,solidx,eta,imom,wrad,angvel,dt,totalmass,
&          nplatelet,pm,Rt,Lmom,Pmom,Ibody,invIbody,omegaMat,vc
    ,
&          niac,pair_i,pair_j,w,indvxdt,totaleta,gammadot,
    exdvxdt)
      integer ntotal,nprobe,nspart,itimestep,nvirt,nentrained,
&          nplatelet,pm,niac
      double precision ap,imom,dt,totalmass,totaleta,gammadot
      REAL (KIND=8), ALLOCATABLE :: mass(:),x(:,:),vx(:,:),rho
     (:),
&          p(:),hsml(:),partdvxdt(:,:),solidx(:,:),eta(:),wrad
    (:),
&          angvel(:),Rt(:,:,:),Lmom(:,:),Pmom(:,:),Ibody
    (:,:,:),
&          invIbody(:,:,:),omegaMat(:,:,:),vc(:,:),w(:),
&          indvxdt(:,:),exdvxdt(:,:)
      INTEGER, ALLOCATABLE :: itype(:), spartent(:),pair_i(:),
&          pair_j(:)
```

```
          END SUBROUTINE

        END INTERFACE

        ALLOCATE (avdudt(1:maxn),ahdudt(1:maxn),indvxdt(1:3,1:maxn),
       &     ardvxdt(1:3,1:maxn),exdvxdt(1:3,1:maxn),asdvxdt(1:3,1:
          maxn),
       &     ns(1:maxn),partdvxdt(1:3,1:maxn),
       &     pair_i(1:max_interaction),pair_j(1:max_interaction),
       &     w(1:max_interaction),dwdx(1:3,1:max_interaction),
       &     STAT=IERR)

c$OMP PARALLEL
c$OMP do
        do  i=1,ntotal+nspart
          avdudt(i) = 0.
          ahdudt(i) = 0.
          do  d=1,dim
            indvxdt(d,i) = 0.
            ardvxdt(d,i) = 0.
            exdvxdt(d,i) = 0.
            asdvxdt(d,i) = 0.
            partdvxdt(d,i) = 0.
            dvx(d,i) = 0.
          enddo
        enddo
c$OMP enddo
c$OMP END PARALLEL


c---  Positions of virtual (boundary) particles:


        if (virtual_part) then
          nvirt = 0
          call virt_part(itimestep, ntotal,nvirt,hsml,mass,x,vx,
       &       rho,u,p,itype,nspart)
```

```
      endif


c---  Interaction parameters, calculating neighboring particles
c     and optimzing smoothing length

c---  Check SPH flag to see if you have to perform the following
   subroutines
      if (sphon) then
         if (nnps.eq.1) then
c           call direct_find(itimestep, ntotal+nvirt,hsml,
c     &          x,niac,pair_i,pair_j,w,dwdx,ns)
         else if (nnps.eq.2) then
c           write(*,*) 'Linked List'
            call link_list(itimestep, ntotal,hsml(nspart+1),
     &          x,niac,pair_i,pair_j,w,dwdx,ns,itype,nspart)
         endif



c---  Density approximation or change rate

         if (summation_density) then
c           write(*,*) 'Density'
            call sum_density(ntotal+nspart,hsml,mass,niac,pair_i,
     &          pair_j,w,itype,rho,numDens)
         else
            call con_density(ntotal+nspart,mass,niac,pair_i,pair_j,
     &           dwdx,vx, itype,drho,imark)
         endif

      endif

c      write(*,*) 'Finished Density'
```

```
c——   Dynamic viscosity:

c          write(*,*) 'Viscosity'
          if (visc) call viscosity(ntotal,itype,x,rho,eta,niac,
     &       pair_i,pair_j,dwdx,vx,mass,srate,itimestep,nspart,
          gammadot)



        if(sphon) then
c——   Internal forces:
c          write(*,*) 'Internal Forces'
          call int_force(itimestep,dt,ntotal,nspart,hsml,mass,vx,
     &       niac,rho,eta, pair_i,pair_j,dwdx,u,itype,x,t,c,p,
          indvxdt,
     &       tdsdt,du,txx,txy,tyy,km,gm,von,tyx,tdotxx,tdotxy,tdotyy
          ,rxx,
     &       rxy,ryy,tauxx,tauxy,tauyy,rhonot,totaleta,gammadot,
          numDens)


c——   Solid Particles:

        if (spart) call solid_part(x,vx,rho,p,mass,itype,ntotal,
           nprobe,
     &       nspart,itimestep,hsml,partdvxdt,ap,nvirt,spartent,
          nentrained,
     &       solidx,eta,imom,wrad,angvel,dt,totalmass,nplatelet,pm,
          Rt,
     &       Lmom,Pmom,Ibody,invIbody,omegaMat,vc,niac,pair_i,pair_j
          ,
     &       w,indvxdt,totaleta,gammadot,exdvxdt)

c——   External forces:

        if (ex_force) call ext_force(ntotal+nspart+nvirt,mass,x,niac
           ,
```

```
     &                     pair_i,pair_j,itype, hsml, exdvxdt,
        numDens,
     &                     dwdx,p,itimestep)


c      Calculating the neighboring particles and undating HSML

c       if (sle.ne.0) call h_upgrade(dt,ntotal, mass, vx, rho, niac
   ,
c     &                     pair_i, pair_j, dwdx, hsml)

c      Calculating average velocity of each partile for avoiding
   penetration

       if (average_velocity) call av_vel(ntotal+nspart,mass,niac,
          pair_i,
     &                 pair_j,w,vx,rho,av,itype,imark,x,hsml,
        numDens)



c---   Convert velocity, force, and energy to f and dfdt
       if(sphon) then
c$OMP PARALLEL
c$OMP do
       do i=1,ntotal
         if(itype(i).eq.2) then
         do d=1,dim
           dvx(d,i) = indvxdt(d,i) + exdvxdt(d,i)

         enddo
         endif
       enddo
c$OMP enddo
c$OMP end parallel

c---   Update acceleration of solid particles
```

```
      if (spart) then
        do i=1, ntotal
         if(itype(i).eq.11.or.itype(i).eq.12) then
            dvx(1,i) = partdvxdt(1,i)
            dvx(2,i) = partdvxdt(2,i)
            dvx(3,i) = partdvxdt(3,i)
          endif
        enddo
      endif

      end
```

### 7.3.18 solid_part.f

```
      subroutine solid_part(x, vx, rho, p, mass, itype,ntotal,
     &                      nprobe,nspart,itimestep,hsml,
     &                      dvxdt,ap,nvirt,spartent,nentrained,
     &                      solidx,eta,imom,wrad,angvel,dt,
        totalmass,
     &                      nplatelet,pm,Rt,Lmom,Pmom,Ibody,
        invIbody,
     &                      omegaMat,vc,niac,pair_i,pair_j,w,
        indvxdt,
     &                      totaleta,gammadot,exdvxdt)


c
c-----------------------------------------------------------------------

c  Subroutine to calculate position of solid particles within the
c  fluid flow.

c
c     itimestep : Current time step  [in]
c     ntotal : Number of particles  [in]
c     nvirt  : Number of virtual particles  [out]
c     nspart : Number of solid particles    [out]
c     dvxdt  : Acceleration of solid particles  [out]
```

```
c     hsml    : Smoothing Length    [in|out]
c     mass    : Particle masses  [in|out]
c     x       : Coordinates of all particles  [in|out]
c     vx      : Velocities of all particles  [in|out]
c     rho     : Density  [in|out]
c     imark   : Number of interactions with non-fluid particles  [
   in|out]
c     itype   : type of particles  [in|out]

      implicit none
      include 'param.inc'


       INTEGER, ALLOCATABLE :: pair_i(:),pair_j(:),itype(:),ns(:),
     &         spartent(:),topbool(:),bottombool(:)
       REAL (KIND=8), ALLOCATABLE :: x(:,:),vx(:,:),mass(:),rho(:),
     &         p(:),wi(:),vxf(:,:),rhof(:),hsml(:),dwdx(:,:),hv(:),
     &         hvs(:),vp(:,:),dvxdt(:,:),dpdx(:),dpdy(:),w(:),
     &         solidx(:,:),eta(:),fd(:,:),xcent(:),ycent(:),zcent
        (:),
     &         wrad(:),angvel(:),fpotent(:,:),Rt(:,:,:),Lmom(:,:),
     &       Pmom(:,:),Ibody(:,:,:),invIbody(:,:,:),vc(:,:),invI
        (:,:,:),
     &         omega(:,:),omegaMat(:,:,:), alpha(:,:), alphaMat
        (:,:,:),
     &         torquetot(:,:),Inorm(:,:,:),center(:,:),fdxtotal(:),
     &         fdytotal(:),fdztotal(:),rvect(:),acent(:),Rtdot(:,:)
        ,
     &         Rtavg(:,:),indvxdt(:,:),exdvxdt(:,:)
        double precision xl,yl,dx,dy,dz,cdist,cutoff,taux,tauy,
           tauz,imom,
     &         dt,zl,sd,fijc,gammadot
        integer i,ngrab,ntotal, pitype, nvirt, nprobe,mp,np,
     &           op,itimestep,niac,
     &           k,j,sp,d,nprobeiac,
     &           nwedgeiac,nspart,nbreak,l,nentrained,maxp,
        nplatelet,
```

```
     &           ncount,f,pm,mfactor
         character name*14, supp*4, fname*25
       double precision  r, selfdens, ap,
     &                   Re, Cd, cdmax, theta,dist,
     &                   totalmass,ax,ay,
     &                   epsilon,sigma,fpottemp,rcutoff,phi,totaleta



       ALLOCATE (vxf(1:3,1:maxn),
     &     vp(1:3,1:maxn),dpdx(1:maxn),dpdy(1:maxn),ns(1:maxn),
     &     topbool(1:maxn),bottombool(1:maxn),fd(1:3,1:maxn),
     &     xcent(1:nplatelet),ycent(1:nplatelet),zcent(1:nplatelet)
         ,
     &     fpotent(1:3,1:maxn),invI(1:3,1:3,1:nplatelet),
     &     omega(1:3,1:nplatelet),alpha(1:3,1:nplatelet),
     &     alphaMat(1:3,1:3,1:nplatelet),torquetot(1:3,1:nplatelet)
         ,
     &     Inorm(1:3,1:3,1:nplatelet),center(1:3,1:nplatelet),
     &     fdxtotal(1:nplatelet),fdytotal(1:nplatelet),
     &     fdztotal(1:nplatelet),rvect(1:3),acent(1:3),
     &     Rtdot(1:3,1:3),Rtavg(1:3,1:3))


        yl = 150e-9
       xl = 2.05268e-6
       zl = 2.05268e-6

c ---- Initialize average rotation matrix
        do i = 1, 3
         do j = 1, 3
           Rtavg(i,j) = 0.0
         enddo
         enddo

c ---- Initialize solid particle accelerations and fluid velocities
```

```
c$OMP PARALLEL
c$OMP do
        do i = 1,ntotal
         if(itype(i).eq.11.or.itype(i).eq.12) then
         do d = 1, dim
          fpotent(d,i)=0.
          dvxdt(d,i)=0.
         enddo
         endif
        enddo
c$OMP enddo
c$OMP END PARALLEL

c ---  Check to see if any of the solid particles are out of range
    ---
c ---  And if so, stop the simulation

c$OMP PARALLEL
c$OMP do
       do i = 1, ntotal
        if(itype(i).eq.11.or.itype(i).eq.12) then
         if (x(1,i).gt.xl.or.x(1,i).lt.0.or.x(2,i).gt.yl.or.x(2,i).
            lt.0.
     &       or.x(3,i).gt.zl.or.x(3,i).lt.0) then
         print *,' >>> ERROR <<< : Particle out of range'
         print *,'Particle:',i
         print *,'Position:',x(:,i)
         stop
         endif
        endif
       enddo
c$OMP enddo
c$OMP END PARALLEL
```

```
c ***  Determine the position of the center of each platelet
      ncount = 0
       do i=1,ntotal
        if(itype(i).eq.12) then
          ncount = ncount + 1
          xcent(ncount) = x(1,i)
          ycent(ncount) = x(2,i)
          zcent(ncount) = x(3,i)
        endif
       enddo
c$OMP PARALLEL
c$OMP do
       do i = 1, nplatelet
         center(1,i) = xcent(i)
         center(2,i) = ycent(i)
         center(3,i) = zcent(i)
       enddo
c$OMP enddo
c$OMP END PARALLEL



c *** Calculate motion for each platelet ***
c$OMP PARALLEL
c$OMP do PRIVATE(taux,tauy,tauz,i,fpottemp,dist,dx,dy,dz,Rtdot)
      do f=1,nplatelet
c Inititalize torque values
       taux = 0
       tauy = 0
       tauz = 0


        fdxtotal(f) = 0
        fdytotal(f) = 0
        fdztotal(f) = 0
```

```
      do i=1,ntotal
       if(itype(i).eq.11.or.itype(i).eq.12) then
        fd(1,i) = 0
        fd(2,i) = 0
        fd(3,i) = 0
        vxf(1,i) = .0024*(1-((x(2,i)-(2*hsml(i)))/(yl)))
        vxf(2,i) = 0
        vxf(3,i) = 0
        if(nano.eq.2) then
          fd(1,i)=(6*pi*eta(i)*ap*(vxf(1,i)-vx(1,i)))/
     &       ((nspart/nplatelet)**(2./3)) + fpotent(1,i)

          fd(2,i)=(6*pi*eta(i)*ap*(vxf(2,i)-vx(2,i)))/
     &       ((nspart/nplatelet)**(2./3)) + fpotent(2,i)
          fd(3,i)=(6*pi*eta(i)*ap*(vxf(3,i)-vx(3,i)))/
     &       ((nspart/nplatelet)**(2./3)) + fpotent(3,i)
        elseif(nano.eq.1) then
          fd(1,i) = (6*pi*eta(i)*hsml(i)*
     &       (vxf(1,i)-vx(1,i)))/(nspart)
          fd(2,i) = (6*pi*eta(i)*hsml(i)*
     &       (vxf(2,i)-vx(2,i)))/(nspart)
          fd(3,i) = (6*pi*eta(i)*hsml(i)*
     &       (vxf(3,i)-vx(3,i)))/(nspart)
        elseif(nano.eq.3.or.nano.eq.4) then
c ---- Temporarily change the way forces are calculated on NS
          fd(1,i) = mass(i)*(indvxdt(1,i)+exdvxdt(1,i))
          fd(2,i) = mass(i)*(indvxdt(2,i)+exdvxdt(2,i))
          fd(3,i) = mass(i)*(indvxdt(3,i)+exdvxdt(3,i))
        endif

        fdxtotal(f) = fdxtotal(f) + fd(1,i)
        fdytotal(f) = fdytotal(f) + fd(2,i)
        fdztotal(f) = fdztotal(f) + fd(3,i)
```

```
c Calculate the x,y,& z distance between the platelet particle and
    the center.
c Also, calculate the torque values by cross multiplying the
   distance with the applied
c force.
          dx = x(1,i) - xcent(f)
          dy = x(2,i) - ycent(f)
          dz = x(3,i) - zcent(f)

          taux = taux + (dy*fd(3,i) - dz*fd(2,i))
          tauy = tauy + (dz*fd(1,i) - dx*fd(3,i))
          tauz = tauz + (dx*fd(2,i) - dy*fd(1,i))
          endif
         enddo

          torquetot(1,f) = taux
          torquetot(2,f) = tauy
          torquetot(3,f) = tauz


c***     Calculate new linear and angular momentum
          Lmom(1,f) = Lmom(1,f) + taux*dt
          Lmom(2,f) = Lmom(2,f) + tauy*dt
          Lmom(3,f) = Lmom(3,f) + tauz*dt
          Pmom(1,f) = Pmom(1,f) + fdxtotal(f)*dt
          Pmom(2,f) = Pmom(2,f) + fdytotal(f)*dt
          Pmom(3,f) = Pmom(3,f) + fdztotal(f)*dt

c***     Calculate translational velocity for each platelet
          vc(1,f) = Pmom(1,f)/(totalmass/nplatelet)
          vc(2,f) = Pmom(2,f)/(totalmass/nplatelet)
          vc(3,f) = Pmom(3,f)/(totalmass/nplatelet)

c***     Calculate the inverse second moment of inertia tensor
          invI(:,:,f) = matmul(matmul(Rt(:,:,f),invIbody(:,:,f)),
     &                    transpose(Rt(:,:,f)))
```

```
c***     Calculate the second moment of inertia tensor
           Inorm(:,:,f) = matmul(matmul(Rt(:,:,f),Ibody(:,:,f)),
     &                     transpose(Rt(:,:,f)))


c***      Calculate angular velocity vector and matrix
           omega(:,f) = matmul(invI(:,:,f),Lmom(:,f))
           omegaMat(1,1,f) = 0
           omegaMat(1,2,f) = -omega(3,f)
           omegaMat(1,3,f) = omega(2,f)
           omegaMat(2,1,f) = omega(3,f)
           omegaMat(2,2,f) = 0
           omegaMat(2,3,f) = -omega(1,f)
           omegaMat(3,1,f) = -omega(2,f)
           omegaMat(3,2,f) = omega(1,f)
           omegaMat(3,3,f) = 0


c***      Update the rotation matrix
            Rt(:,:,f) = Rt(:,:,f) +
     &             (matmul(omegaMat(:,:,f),Rt(:,:,f))*dt)

c***       Calculate angular acceleration vector and matrix
           alpha(:,f) = matmul(invI(:,:,f),(torquetot(:,f)-
     &     (matmul(omegaMat(:,:,f),matmul(Inorm(:,:,f),omega(:,f)))
          )))
           alphaMat(1,1,f) = 0
           alphaMat(1,2,f) = -alpha(3,f)
           alphaMat(1,3,f) = alpha(2,f)
           alphaMat(2,1,f) = alpha(3,f)
           alphaMat(2,2,f) = 0
           alphaMat(2,3,f) = -alpha(1,f)
           alphaMat(3,1,f) = -alpha(2,f)
           alphaMat(3,2,f) = alpha(1,f)
           alphaMat(3,3,f) = 0
cc          enddo
           enddo
```

```
c$OMP enddo
c$OMP END PARALLEL

        if (mod(itimestep,save_step).eq.0) then
            open(22,file='NS_ang_acc.txt')
            write(22,*) alpha(1,1), alpha(2,1),alpha(3,1)
        endif
        if (mod(itimestep,print_step).eq.0) then
            write(*,*) alpha(1,1), alpha(2,1), alpha(3,1)
        endif


c***     Calculate the acceleration for each platelet particle
c$OMP PARALLEL
c$OMP do PRIVATE(i,rvect,dx,acent)
         do f = 1, nplatelet
           do i=1,ntotal
            if(itype(i).eq.11.or.itype(i).eq.12) then
c-           First calculate the translational acceleration
             dvxdt(1,i) = fdxtotal(f)/(totalmass/nplatelet)
             dvxdt(2,i) = fdytotal(f)/(totalmass/nplatelet)
             dvxdt(3,i) = fdztotal(f)/(totalmass/nplatelet)
c-           Now add the angular acceleration
c       *** Calculate rvect differently if platelet split ***
            rvect(1) = x(1,i)-center(1,f)
            rvect(2) = x(2,i)-center(2,f)
            rvect(3) = x(3,i)-center(3,f)

              dvxdt(:,i) = dvxdt(:,i) +
     &              matmul(alphaMat(:,:,f),rvect)

c-           Now add the centripetal acceleration
             acent = matmul(omegaMat(:,:,f),rvect)
             dvxdt(:,i) = dvxdt(:,i) +
     &           matmul(omegaMat(:,:,f),acent)
            endif
           enddo
```

```
          enddo
c$OMP enddo
c$OMP END PARALLEL

          fijc = 0.0
c ***   Calculate correction to stress tensor due to constraint
c       forces
          if(mod(itimestep,save_step).eq.0) then
          do i = 1, ntotal
            if(itype(i).eq.11.or.itype(i).eq.12) then
              fijc = fijc + ((mass(i)*dvxdt(1,i))-(mass(i)*
     &                       (indvxdt(1,i)+exdvxdt(1,i))))
            endif
          enddo
          write(*,*) 'Correction due to constraint forces:', fijc
           totaleta = (totaleta + fijc)/(gammadot)
          open(76,file='viscosity2.txt')
          write(76,*) itimestep,totaleta
          write(*,*) 'gammadot:',gammadot
          write(*,*) 'Total Viscosity:',totaleta
          endif

        end
```

### 7.3.19 time_integration.f

```
       subroutine time_integration(x,vx, mass, rho, p, u, c, s, e,
          itype,
     &             hsml,ntotal, maxtimestep, dt,nstart,endtime,
          endgrab,xl,
     &             yl,km,gm,von,dvx,nspart,ap,imom,totalmass,
          nplatelet,pm,
     &             Rt,Lmom,Pmom,Ibody,invIbody,nvirt,zl,numDens)

c
```

```
c       x— coordinates of particles   [input/output]
c       vx— velocities of particles  [input/output]
c       mass— mass of particles  [input]
c       rho— dnesities of particles [input/output]
c       p— pressure  of particles  [input/output]
c       u— internal energy of particles  [input/output]
c       c— sound velocity of particles  [output]
c       s— entropy of particles, not used here   [output]
c       e— total energy of particles   [output]
c       itype— types of particles   [input]
c            =1   ideal gas
c            =2   water
c            =3   tnt
c       hsml— smoothing lengths of particles [input/output]
c       ntotal— total particle number   [input]
c       maxtimestep— maximum timesteps  [input]
c       dt— timestep  [input]


        implicit none
        include 'param.inc'

        INTEGER, ALLOCATABLE :: itype(:), imark(:), spartent(:)
        integer ntotal, maxtimestep,ngrab,nvirt,nprobe,ierr,stat,num
           ,
     &       stotal,nspart,nentrained,nplatelet,pm
        REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:), rho
           (:),
     &       p(:),u(:), c(:), s(:), e(:), hsml(:), txx(:), tyy(:),
          txy(:),
     &       tyx(:),tdotxx(:),tdotyy(:),tdotxy(:),rxx(:),rxy(:),ryy
          (:),
     &       tauxx(:),tauxy(:),tauyy(:),tempx(:,:),solidx(:,:),
          angvel(:),
     &       wrad(:),Rt(:,:,:),Lmom(:,:),Pmom(:,:),Ibody(:,:,:),
     &       invIbody(:,:,:),omegaMat(:,:,:),vc(:,:),rhonot(:),
          srate(:),
```

```
     &       eta(:),numDens(:)
      double precision dt, top, temp_x, temp_y, rtemp_x, rtemp_y,
         r,
     &       theta, dist, midptx, midpty,xlim,ylim,ap
      integer i, j, k, itimestep, d, current_ts, nstart,endgrab,
         nghost,
     &       nboundary, file, sum, incount
        REAL (KIND=8), ALLOCATABLE :: x_min(:,:), v_min(:,:),
           u_min(:),
     &       rho_min(:),dx(:,:), dvx(:,:), du(:), drho(:), av
         (:,:),
     &       ds(:), t(:),tdsdt(:),psitype(:),km(:),gm(:),von(:)
c       double precision  x_min(3, maxn), v_min(3, maxn), u_min(
   maxn),
c     &       rho_min(maxn), dx(3,maxn), dvx(3, maxn), du(maxn),
c     &       drho(maxn),  av(3, maxn), ds(maxn),
c     &       t(maxn), tdsdt(maxn)
      double precision  time, temp_rho, temp_u,endtime,xl,yl,
         delvx,
     &       delvy, delx, dely, timestep,mag,phi,hx,hy,imom,
     &       totalmass,zl
      character one*1,two*2,three*3, four*4, five*5,name1*11,
     &         name2*12,name3*13,name4*14,name5*15, path*80

      INTERFACE
        SUBROUTINE probe_part(x, vx, rho, p, mass, itype,
     +                        ntotal,nprobe,nvirt,nspart,ngrab,
     +                        itimestep,hsml,txx,txy,tyy,imark)
          REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), rho(:),
     +                        p(:),mass(:),hsml(:),txx(:),tyy(:),
        txy(:)
          INTEGER, ALLOCATABLE :: itype(:), imark(:)
          integer :: ntotal, nvirt, ngrab, nspart
          END SUBROUTINE probe_part

          SUBROUTINE output_vtu(x, vx, rho, p, mass, itype, ntotal
             ,
```

```
+                          ngrab, imark, von, nvirt, nspart,eta,
   srate,
+                          numDens)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), rho(:), p
          (:),
+                           mass(:), von(:),rhonot(:),eta(:),
   srate(:),
+                           numDens(:)
      INTEGER, ALLOCATABLE :: itype(:), imark(:)
      integer :: ntotal, ngrab, nvirt, nspart
      END SUBROUTINE

      SUBROUTINE output_recovery(x, vx, rho, p, mass, itype,
          ntotal,
+               nspart,hsml, itimestep,time,ngrab,dvx,nvirt)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), rho(:), p
          (:),
+                                 mass(:), hsml(:), dvx(:,:)
      INTEGER, ALLOCATABLE :: itype(:)
      integer :: ntotal, nvirt, itimestep,ngrab,nspart
      double precision time
      END SUBROUTINE

      SUBROUTINE single_step(itimestep,dt,ntotal,hsml,mass,x,
          vx,u,s,
+       rho,p,t,tdsdt,dx,dvx,du,ds,drho,itype,av,nvirt,xl,yl,
   nghost,
+       v_min,txx,txy,tyy,imark,km,gm,von,tyx,tdotxx,tdotxy,
   tdotyy,
+       rxx,rxy,ryy,tauxx,tauxy,tauyy,nspart,ap,spartent,
   nentrained,
&       solidx,imom,wrad,angvel,totalmass,nplatelet,pm,Rt,Lmom
   ,Pmom,
&       Ibody,invIbody,omegaMat,vc,rhonot,srate,eta,numDens)
      REAL (KIND=8), ALLOCATABLE :: x(:,:), vx(:,:), mass(:),
          rho(:),
```

```
     &       p(:),u(:), s(:), hsml(:), t(:), tdsdt(:), dx(:,:), dvx
         (:,:),
     &       du(:),ds(:),drho(:),av(:,:),v_min(:,:),txx(:), tyy(:),
         txy(:),
     &       km(:),gm(:), von(:),tyx(:),tdotxx(:),tdotyy(:),tdotxy
         (:),
     &       rxx(:),rxy(:),ryy(:),tauxx(:),tauxy(:),tauyy(:),solidx
         (:,:),
     &       wrad(:),angvel(:),Rt(:,:,:),Lmom(:,:),Pmom(:,:),Ibody
         (:,:,:),
     &       invIbody(:,:,:),omegaMat(:,:,:),vc(:,:),rhonot(:),srate
         (:),
     &       eta(:),numDens(:)
           INTEGER, ALLOCATABLE :: itype(:), imark(:), spartent(:)
           integer itimestep, ntotal, nghost, nvirt, nspart,
              nentrained,
     &       nplatelet,pm
           double precision dt, xl, yl, ap,imom,totalmass
           END SUBROUTINE
       END INTERFACE

       ALLOCATE (av(1:3,1:maxn),u_min(1:maxn),v_min(1:3,1:maxn),
     &    drho(1:maxn),rho_min(1:maxn),tempx(1:2,1:maxn),spartent
         (1:maxn)
     &    ,solidx(1:2,1:maxn),angvel(1:100),wrad(1:100),
     &    omegaMat(1:3,1:3,1:nplatelet),vc(1:3,1:nplatelet),
     &    rhonot(1:maxn),srate(1:maxn),eta(1:maxn),STAT=IERR)


c ---  Initialize angular velocity matrix and translational
     velocity
c       for each platelet
        do i = 1, nplatelet
          do j = 1, 3
            vc(j,i) = 0
          enddo
        enddo
```

```
      do i = 1, nplatelet
        do j = 1, 3
          do k = 1, 3
            omegaMat(j,k,i) = 0
          enddo
        enddo
      enddo


      do i = 1, ntotal+nspart
        do d = 1, dim
          av(d, i) = 0.
        enddo
      enddo



      ngrab = 0
      time = 0
      current_ts = 0
c *** Initialize angular velocity and acceleration of platelets
      do i = 1, nplatelet
       angvel(i) = 0.
       wrad(i) = 0.
      enddo

      do itimestep = nstart+1, nstart+maxtimestep

        current_ts=current_ts+1
        if (mod(itimestep,print_step).eq.0) then
         write(*,*)'-----------------------------------------------
            '
         write(*,*)'  current number of time step =',
     &            itimestep,'     current time=', real(endtime+time
        +dt)
```

```
          write(*,*)'-----------------------------------------------
             '
          endif



c     If not first time step, then update thermal energy, density
   and
c     velocity half a time step
c       if(sphon) then
          if (itimestep .ne. 1) then
c$OMP PARALLEL Private(temp_rho)
c$OMP do
            do i = 1, ntotal
             if(itype(i).eq.2.or.itype(i).eq.11.or.itype(i).eq.12)
                then
c             if(itype(i).eq.11.or.itype(i).eq.12) then
             if(.not.summation_density) then
               rho_min(i) = rho(i)
               temp_rho = 0.
               rho(i) = rho(i) + (dt/2.)*(drho(i)+temp_rho)
              endif
             do d = 1, dim
               v_min(d, i) = vx(d, i)
               vx(d, i) = vx(d, i) + (dt/2.)*dvx(d, i)
             enddo
             endif
           enddo
c$OMP enddo
c$OMP END PARALLEL
           endif


c---  Definition of variables out of the function vector:
```

```
      call single_step(itimestep, dt, ntotal, hsml, mass, x, vx,
          u, s,
     &       rho, p, t, tdsdt, dx, dvx, du, ds, drho,itype, av,
     nvirt,xl,
     &       yl,nghost,v_min,txx,txy,tyy,imark,km,gm,von,tyx,
     tdotxx,
     &       tdotxy,tdotyy,rxx,rxy,ryy,tauxx,tauxy,tauyy,nspart,ap
     ,
     &       spartent,nentrained,solidx,imom,wrad,angvel,totalmass
     ,
     &       nplatelet,pm,Rt,Lmom,Pmom,Ibody,invIbody,omegaMat,vc,
     &       rhonot,srate,eta,numDens)



c       if(sphon) then
        if (itimestep .eq. 1) then
c$OMP PARALLEL PRIVATE(temp_rho)
c$OMP do
          do i=1,ntotal
           if(itype(i).eq.2.or.itype(i).eq.11.or.itype(i).eq.12)
              then
c           if(itype(i).eq.11.or.itype(i).eq.12) then
           if(.not.summation_density) then
            temp_rho = 0.
            rho(i) = rho(i) + (dt/2.)*(drho(i)+temp_rho)
           endif
            do d = 1, dim
             if(itype(i).eq.2) then
              vx(d, i) = vx(d, i) + (dt/2.) * dvx(d, i) + av(d, i)
             else
               vx(d,i) = vx(d,i) + (dt/2) * dvx(d,i)
             endif
               x(d, i) = x(d, i) + dt * vx(d, i)
            enddo
            endif
          enddo
```

```
c$OMP enddo
c$OMP END PARALLEL
        else
c$OMP PARALLEL PRIVATE(temp_rho)
c$OMP do
          do i=1,ntotal
           if(itype(i).eq.2.or.itype(i).eq.11.or.itype(i).eq.12)
              then
c          if(itype(i).eq.11.or.itype(i).eq.12) then
           if(.not.summation_density) then
             temp_rho = 0.
             rho(i) = rho_min(i) + dt*(drho(i)+temp_rho)
           endif
            do d = 1, dim
             if(itype(i).eq.2) then
              vx(d, i) = v_min(d, i) + dt * dvx(d, i) + av(d, i)
             else
              vx(d,i) = v_min(d,i) + (dt) * dvx(d,i)
             endif
               x(d, i) = x(d, i) + dt * vx(d, i)
            enddo
            endif
          enddo
c$OMP enddo
c$OMP END PARALLEL
       endif



c*** Enforce Periodic BC's for SPH Particles
c--  Don't worry about interaction periodicity yet
c$OMP PARALLEL
c$OMP do
         do i = 1, ntotal
          if(itype(i).eq.2.or.itype(i).eq.11.or.itype(i).eq.12)
             then
          if(x(1,i).ge.(xl+(2*hsml(1)))) then
```

```
            x(1,i) = x(1,i) - xl
          elseif(x(1,i).le.(2*hsml(1))) then
            x(1,i) = x(1,i) + xl
          elseif(x(3,i).ge.(xl+(2*hsml(1)))) then
            x(3,i) = x(3,i) - xl
          elseif(x(3,i).le.(2*hsml(1))) then
            x(3,i) = x(3,i) + xl
           endif
         endif
         enddo
c$OMP enddo
c$OMP END PARALLEL

        time = time + dt



c---  Positions of probe particles:

       nprobe = 0.


c--- Output VTU files for visualization later in ParaView
            if (mod(itimestep,save_step).eq.0) then
            ngrab =  ngrab + 1
            if (probe_particle) then
              call probe_part(x, vx, rho, p, mass, itype,
     +                  ntotal,nprobe,nvirt,nspart,ngrab+endgrab,
     +                  itimestep,hsml,txx,txy,tyy,imark)
            endif
            call output_vtu(x, vx, rho, p, mass, itype,
     +                ntotal+nvirt+nspart+nprobe,ngrab+endgrab,
        imark,von,
     +                nvirt,nspart,eta,srate,numDens)
c       Calculate velocity of center particle in NS
c$OMP PARALLEL
c$OMP do
```

```
          do i = 1, ntotal
           if(itype(i).eq.12) then
            open(92,file='velNS.txt')
            write(92,*) vx(1,i), vx(2,i), vx(3,i)
           endif
          enddo
c$OMP enddo
c$OMP END PARALLEL


            endif

        if (mod(itimestep,rec_step).eq.0) then
          if(output_rec) then
           call output_recovery(x, vx, rho, p, mass, itype,
     +          ntotal,nspart,hsml,itimestep,time+endtime,ngrab+
        endgrab,
     +               dvx,nvirt)
          endif
        endif


       enddo

         DEALLOCATE(av,u_min,v_min)

       nstart= itimestep-1
       endtime = time
       endgrab = endgrab + ngrab



       end
```

### 7.3.20 viscosity.f

```
       subroutine viscosity(ntotal,itype,x,rho,eta,niac,pair_i,
          pair_j,
```

```
     &                    dwdx,vx,mass,srate,itimestep,nspart,gammadot
        )

c
     ------------------------------------------------------------------

c    Subroutine to define the fluid particle viscosity

c      ntotal  : Number of particles   [in]
c      itype    : Type of particle    [in]
c      x       : Coordinates of all particles [in]
c      rho     : Density [in]
c      eta     : Dynamic viscosity   [out]

       implicit none
       include 'param.inc'

       INTEGER, ALLOCATABLE :: itype(:),pair_i(:),pair_j(:)
       integer ntotal,i, IERR,niac,k,j,d,e,itimestep,nspart,
          ntotalreal
       REAL (KIND=8), ALLOCATABLE :: x(:,:), rho(:), eta(:),dwdx
          (:,:),
      &                    vx(:,:),mass(:),srate(:),srateten(:,:,:),
      &                    temprate(:)
       double precision gammadot

       Allocate (srateten(1:3,1:3,1:maxn),temprate(1:maxn))

       gammadot = 0.0
       ntotalreal = 0

       if(var_visc) then
c *** Initialize Shear Rate Tensor
c$OMP PARALLEL
c$OMP do
        do i = 1, ntotal
         do d = 1, 3
```

```
          do e = 1, 3
            srateten(d,e,i) = 0.0
          enddo
        enddo
        temprate(i) = 0.0
        enddo
c$OMP enddo
c$OMP END PARALLEL


c         write(*,*) 'Calculate local shear rate tensor'
c *** Calculate the local shear rate tensor for each particle
c$OMP PARALLEL PRIVATE(i,j)
c$OMP do
      do k = 1, niac
         i = pair_i(k)
         j = pair_j(k)
         if(itype(i).ne.4.and.itype(j).ne.4) then
         do d = 1, 3
           do e = 1, 3
              srateten(d,e,i) = srateten(d,e,i) + ((dwdx(d,k)*mass
                 (j)*
     &             (vx(e,i)-vx(e,j)))/rho(j))

              srateten(d,e,j) = srateten(d,e,j) + ((dwdx(d,k)*mass
                 (i)*
     &             (vx(e,i)-vx(e,j)))/rho(i))
           enddo
         enddo
         endif
       enddo
c$OMP enddo
c$OMP END PARALLEL



c *** Calculate the local magnitude shear rate and viscosity
```

```
c$OMP PARALLEL
c$OMP do
        do i = 1, ntotal
          if(itype(i).ne.4) then
           do d = 1, 3
             do e = 1, 3
               temprate(i) = temprate(i) + (srateten(d,e,i))**2
             enddo
           enddo
           srate(i) = sqrt(temprate(i)/2)
             if(srate(i).lt.0) then
                  write(*,*)'Particle ',i, ' has a negative
     &              shear rate of:',srate(i)
                  eta(i) = 0.13
             else
                  eta(i) = ((-2e-11)*srate(i)**2) -
     &               ((2e-7)*srate(i)) + 0.127
             endif
           endif
         enddo
c$OMP enddo
c$OMP END PARALLEL

         do i = 1, ntotal
           if(itype(i).eq.2) then
             gammadot = gammadot + srateten(2,1,i)
             ntotalreal = ntotalreal + 1
           endif
         enddo

        gammadot = gammadot/ntotalreal

        else

       endif
```

```
end
```